DISSERTATION

MULTI-STREAM DEEP LEARNING FOR ISOLATED SIGN LANGUAGE RECOGNITION IN VIDEOS

Submitted by

Muhammad H. Alsharif

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2025

Doctoral Committee:

Advisor: Charles Anderson

Michael Kirby
Nathaniel Blanchard
Christopher Peterson

ABSTRACT

MULTI-STREAM DEEP LEARNING FOR ISOLATED SIGN LANGUAGE RECOGNITION IN VIDEOS

Isolated sign language recognition is the task of identifying signs performed in isolation across multiple frames in a video. Advances in this field have significant implications, such as improving visual communication between humans and machines and bridging the communication gap between deaf and hearing individuals. However, practical applications of this domain have been limited by two key challenges: the computational complexity of current models and the limited availability of training data for many vocabularies in sign languages. This dissertation addresses these challenges driven by improving recognition accuracy and computational efficiency.

3D convolutional models with RGB and optical flow inputs have been widely utilized in state-of-the-art methods for action recognition. Despite their significant computational costs, a systematic evaluation of their contribution to sign recognition has been limited. We first evaluate the effectiveness of 3D convolutional networks, showing that they significantly outperform their 2D counterparts on several sign recognition datasets, even when compared to a deeper 2D architecture. Additionally, this research challenges conventual assumptions about optical flow, demonstrating through ablation studies that its primary value lies in masking irrelevant (static) regions rather than improving the learning of motion patterns for sign recognition.

In addition to RGB and optical flow, this work investigates skeleton-based sign language recognition using recurrent, transformer, and spatiotemporal convolutional graph networks. Our experimental results demonstrate the importance of the spatiotemporal sparse graph representation of skeleton data (coordinates of body and hand joints) in improving accuracy

and interpretability through edge importance weighting. To address the limited number of training data for many signs, we propose a coarse-to-fine transfer learning approach to adapt spatiotemporal features learned from large action recognition and Turkish sign language datasets to American sign language datasets. This approach results in significant improvement for multiple modalities and benchmarks.

To combine different models in a multi-stream network, we propose several methods for fusing the stream outputs before and after classification. To find the best combination of models using RGB, optical flow, or skeleton as input modalities, we train and evaluate all possible combinations in two- and three-stream networks on three sign recognition datasets. Our findings show that combining RGB and skeleton-based streams provides the most significant gain over the RGB baseline, due to greater diversity in stream predictions. In contrast, combining RGB and optical flow-based streams significantly increases the computational cost, due to optical flow extraction, without improving accuracy over two RGB streams.

Our two- and three-stream networks, using only RGB and skeleton data as input modalities, achieve new state-of-the-art accuracy on the two largest ASL video datasets, which include 1000 and 2000 signs. Our approach achieves over 90% top-5 recognition accuracy on all benchmarks while significantly reducing computational costs compared to state-of-the-art methods. These findings facilitate real-time applications on mobile devices aimed at improving convenience in the daily lives of deaf individuals and helping to overcome communication barriers.

ACKNOWLEDGEMENTS

I am deeply grateful to the Almighty for granting me the strength, guidance, and perseverance to complete this work. I am very thankful to my parents for their unwavering support and sacrifices, which have made this journey possible.

I sincerely thank my advisor, Professor Chuck Anderson, whose encouragement, support, and constructive feedback were instrumental throughout my Ph.D. journey. I also appreciate my committee members for their time and engagement. The computational resources and GPUs provided by the department's Falcon Cluster have been essential to the completion of my experiments, and I am grateful for their availability. Finally, I acknowledge the researchers and engineers whose pioneering work in deep learning has laid the foundation for progress in this field.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The ability of machines to recognize and interpret human actions is central to many cutting-edge technologies, including autonomous systems, healthcare monitoring, and interactive communication tools. Action recognition in videos aims to classify the action performed by a human—such as running, walking, waving, or riding a bike—across multiple frames by analyzing both spatial and temporal information. To achieve this, deep learning models originally developed for image recognition have been adapted for action recognition by incorporating a temporal dimension into 2D spatial filters [1–4].

However, a close examination of the accuracy achieved by the 3D convolutional models on action recognition benchmarks [5–8] reveals a marginal improvement over their 2D counterparts, as observed in [7] and [9]. This calls into question the merit of using 3D convolutional models over 2D for action recognition tasks, as the former is more computationally intensive. One possible explanation for the limited improvement is that many actions in these datasets are associated with certain objects or backgrounds, thus providing sufficient clues about the action to the model without requiring it to learn the motion pattern.

Additionally, the use of two-stream networks—one for processing the RGB input and the other for processing the optical flow frames extracted from the RGB input—has been widely adopted in multiple state-of-the-art methods (e.g., [2, 10–13]). This approach relies on the hypothesis that the optical flow stream is better at capturing motion, thus providing complementary information to the RGB stream intended for capturing appearance features [14]. While this approach improves accuracy over a single-stream baseline, no direct comparison is provided to a two-stream network where both streams process the same RGB input. This is an important consideration because introducing more modalities into a network architecture adds significant computational overhead, making the model less practical for mobile devices and real-time applications.

In this dissertation, we approach isolated sign language recognition as a subcategory of action recognition, where actions are characterized by certain hand configurations and motion patterns. This perspective offers a more rigorous testbed for the spatiotemporal models compared to general action recognition datasets, as the background of the signer and nearby objects are not correlated with the performed sign. Moreover, motion constitutes a key discriminative feature of signs, thus enabling a better evaluation of the contribution of 3D convolutional models and optical flow streams in multi-stream networks.

While sign language recognition serves as a valuable testbed for advancing spatiotemporal models and visual communication between humans and machines, its practical and societal implications are even more compelling. There are more than 70 million deaf people worldwide [15] and 430 million experiencing disabling hearing loss [16]. Automated sign language recognition can bridge the communication gap between deaf and hearing individuals, similar to how real-time translation applications facilitate communication between speakers of different languages.

In the following sections, the problem of sign language recognition is first defined, followed by the research questions and an overview of our approach. Subsequently, we introduce the sign language datasets and their challenges. This chapter concludes with a summary of contributions and an outline of the remaining chapters.

1.1 Problem Definition and Scope

Sign language recognition (SLR) can be broadly categorized into two forms: isolated recognition and continuous recognition. Isolated SLR focuses on identifying a single sign, corresponding to a specific word, in a video clip containing only that sign. In contrast, continuous SLR seeks to interpret a sequence of connected signs in a video to generate the corresponding sentence in a spoken language [17]. This dissertation focuses on the first problem of isolated SLR as a special case of action recognition. Nonetheless, advancements

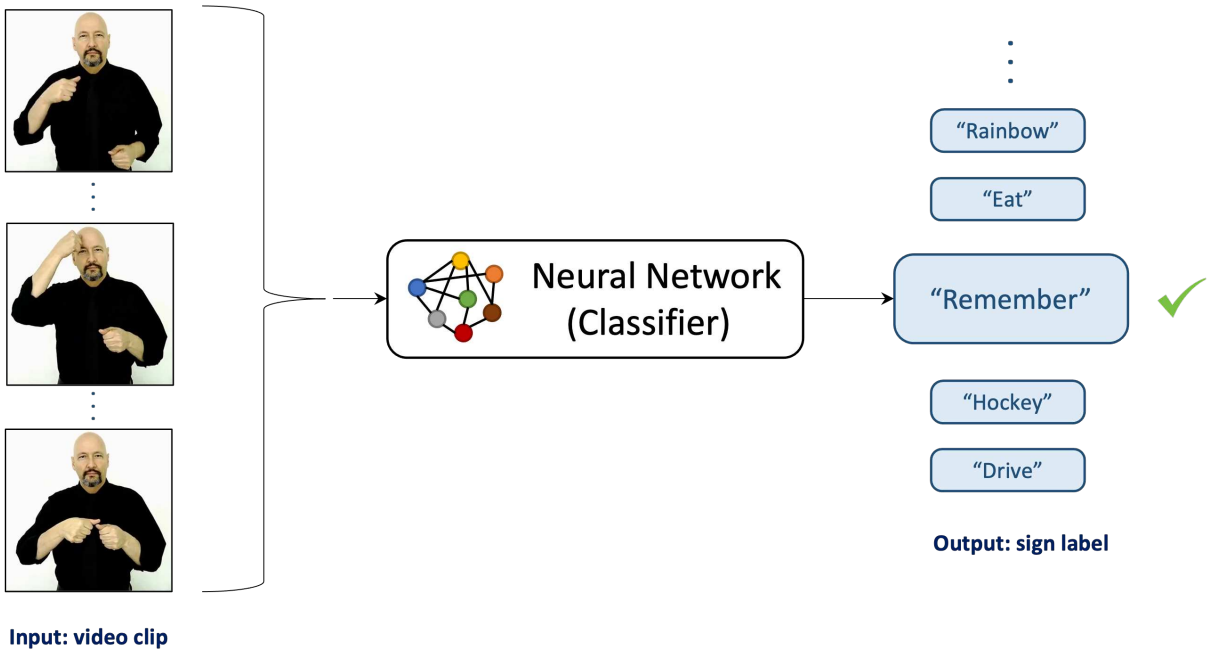


Figure 1.1: Illustration of isolated sign language recognition as a classification problem. The neural network is trained to predict the correct label (class) for any given input clip.

in isolated SLR research can be transferred to the continuous SLR domain since the latter involves recognizing individual words as part of generating a coherent sentence.

Isolated sign language recognition is a classification problem where the input to the model is a video, typically less than 10 seconds long, containing a single sign performed once or multiple times across the frames. The output is a probability vector, with each element representing the confidence that the input video belongs to a specific class. The predicted label corresponds to the class with the highest probability among all classes (Figure 1.1). Each isolated SLR video dataset has a defined number of classes (labels) ranging from 100 to 2000 classes.

From a machine learning perspective, isolated SLR is a supervised learning problem in which the model is trained to correctly classify the input videos in a training set by adjusting its weights to ensure its predicted labels match the ground truth labels. The model's generalization accuracy is then evaluated on unseen samples in the test set by comparing its

predictions to the ground truth labels and calculating the proportion of the correct classifications.

1.2 Research Questions

While several advances in action recognition have proven useful for isolated sign language recognition, some challenges remain unresolved, including (i) the limited amount of training videos relative to the size of the sign language vocabulary, and (ii) balancing the trade-off between computational efficiency and accuracy. Although incorporating additional modalities with hand-crafted features, such as optical flow, can help address the first limitation, it hinders the second challenge by increasing the computational overhead of the network, making it less practical for real-time usage with limited computational resources. Motivated by the need to maximize recognition accuracy while minimizing computational cost, this dissertation investigates the following key research questions.

1. How important is learning temporal features across all layers of a 3D convolutional network (slow fusion) for sign language recognition compared to aggregating frame-level features only in the final layer of a 2D convolutional network (late fusion)? This is motivated by the observation that, while a 3D convolutional network is more computationally intensive than its 2D counterpart, it has shown limited gain in action recognition benchmarks such as the Kinetics dataset [8].

2. Does the inclusion of optical flow modality in a multi-stream network justify the added computational cost compared to the inclusion of other modalities, such as RGB and skeleton? Is the utility of optical flow in sign language recognition primarily due to capturing of motion, or does it stem from other factors?

3. What is the optimal strategy for fusing stream outputs, and which combination of streams (modalities) consistently results in the best improvement over the baseline? Are certain classes of signs recognized significantly better with a specific modality?

To address the limited number of training videos per class, we propose a coarse-to-fine training for all streams in the network, where each model is first trained on an action recognition dataset, then on a sign dataset of a different language, and finally on the target sign language dataset. The underlying hypothesis is that this progression enables the model to first learn fundamental spatial and temporal features useful for general action recognition tasks, then adapt these spatiotemporal features to higher-level, domain-specific patterns in gesture recognition, and finally fine-tune to capture the specific nuances of the target sign language dataset.

To address the aforementioned research questions, we first conduct comprehensive experiments to: (i) analyze the impact of the slow fusion (3D convolutions) compared to the late fusion of temporal features, and ii) analyze the effect of different components of optical flow on accuracy to show its primary utility in isolated SLR is best understood as masking out irrelevant features from the input, rather than in learning motion patterns, as commonly assumed. We refer to this utility as the masking hypothesis. Based on these insights, we construct, train, and evaluate all possible combinations of two- and three-stream networks on three sign language datasets: AUTSL, MSASL, and WLASL utilizing RGB, optical flow, and skeleton as input modalities. We systematically compare their accuracy gains over a single-stream baseline, aiming to identify the right balance between accuracy and computational cost.

1.3 Sign Language Datasets

Sign language researchers and linguists have produced several sign language datasets in the past two decades for different spoken languages. Each dataset differs in the number of signers, vocabularies (classes), video samples per vocabulary, and whether the recording was done using a single or multiple cameras [18], or special equipment such as Kinect or depth sensors [19–21]. To ease the recognition task, some of these datasets utilize plain backgrounds or have the signers wear colored gloves [22] to facilitate tracking of hands and fingers. In

Table 1.1: Comparison of isolated sign language recognition datasets

Dataset	Country	Classes	Signers	Samples	Year
Boston ASLLVD [23]	USA	2742	6	9794	2008
DEVISIGN-L [24]	China	2000	8	24,000	2014
SLR500 [25]	China	500	50	125,000	2018
Word-Level ASL [26]	USA	2000	119	21,083	2020
Microsoft ASL [27]	USA	1000	222	25,513	2020
AUTSL [28]	Turkey	226	43	38,336	2020

this work, we focus on datasets with a single RGB camera without using special equipment or gloves so that our approach could be extended to mobile devices for communicating on the spot with humans or machines.

Table 1.1 lists the publicly available sign language datasets at the time of this writing. Not all sign datasets are suitable for deep learning approaches, as some are either not large enough or lack sufficient variety in their signs or signers. The following criteria guided our selection of sign language datasets for this research.

1. The videos should be monocular (recorded using a single RGB camera) and no special gloves or equipment. This excludes datasets whose videos were recorded with other sensors such as depth cameras or lidar.

2. The dataset should contain enough samples per class to avoid fast overfitting during training.

3. The videos should be recorded as isolated signs (words), not continuous sentences.

4. The dataset should be signer-independent: each sign should be performed by various signers, and the signers in the training, validation, and test sets are mutually exclusive. This reduces the likelihood of associating the recognition of a sign with certain individuals and allows us to test if the model can generalize to unseen individuals performing the same sign.

All datasets listed in Table 1.1 satisfy the first criterion. However, Boston ALLVD does not provide a sufficient number of training samples relative to the number of classes. Both ALLVD and Devisign datasets have relatively few signers and fail to meet the signer-independence requirement. In contrast, Word-Level ASL (WLASL), Microsoft ASL (MSASL), AUTSL, and SLR500 datasets sataisfy the four criteria.

Among these, WLASL and MSASL are the most challenging due to their relatively lower number of samples per sign compared to AUTSL and SLR500. While SLR500 contains the largest number of samples, we exclude it from our experiments for two reasons. First, its signs are recorded in a controlled environment with identical backgrounds, which limits its applicability to real-world scenarios. Second, its size requires substantial resources for training in a timely manner, which is beyond the scope of this work. In this dissertation, we use the AUTSL, MSASL, and WLASL datasets.

1.3.1 AUTSL

AUTSL (Ankara University Turkish Sign Language) [28] is a large-scale dataset for Turkish isolated sign language recognition. The authors collaborated with Turkish sign language instructors to select frequently used signs containing a variety of hand shapes and motions. The signs in this dataset were performed by 43 different signers, including instructors, translators, students, and trainees. The dataset includes 20 distinct backgrounds, covering various indoor and outdoor scenes and lighting conditions to provide realistic scenarios. To make it more challenging, the authors included very similar signs that differ only in one aspect, such as hand shape, orientation, position, or motion.

The authors recorded the videos in the dataset using Microsoft Kinect v2 and applied clipping and resizing to provide 512x512 resolution for all videos. The recordings have a frame rate of 30 FPS, and video lengths ranging between 1 and 3 seconds. The dataset is split by the authors into 72% of the samples for training, 13% for the validation, and 15%

for the testing. Among the three datasets discussed in this section, this dataset is the most carefully curated and has the best recording quality.

1.3.2 MS-ASL

Microsoft American Sign Language (MS-ASL) [27] is a large-scale ASL benchmark with 1000 vocabularies (words) released by Microsoft Research in 2020. The videos in this dataset were captured and uploaded to YouTube by over 200 ASL professionals, teachers, and students. These videos were recorded in various backgrounds, views, positions, and lighting which makes it a bit more challenging. Nevertheless, this formed a dataset with realistic data sources suitable for recognition systems designed to perform well in realistic circumstances (Figure 1.2 a).

The videos in this dataset were classified and labeled by the authors [29] based on the text detected in the video using OCR, or the title of the video. Face recognition and bounding boxes of the signer were detected and used to split long videos that included multiple signers. Samples with more than 6-8 seconds were manually trimmed so they are not too long for recognition tasks. The labels, bounding boxes, starting and end times of each sign instance, and other metadata for all videos were made available as JSON files by the authors.

The authors split the dataset into training, validation, and test sets such that the signers in these partitions are mutually exclusive, and therefore the signer independence is fulfilled. Out of the 222 total signers, 165 signers were assigned to the training set, 37 to the validation set, and 20 to the test set [27] or, in other words, a distribution of 74%, 17%, and 9%, respectively. Since the frequency of samples by each signer in the corpus is very diverse (ranges from 1 to 1000), the distribution of samples among the three partitions is not the same as the respective distribution of the signers.

The MSASL dataset has been further divided into four benchmarks (subsets) called MSASL-100, MSASL-200, MSASL-500, and MSASL-1000. The number after 'MSASL' refers to the number of classes in that subset. MSASL-100 has the most frequent 100 signs, and

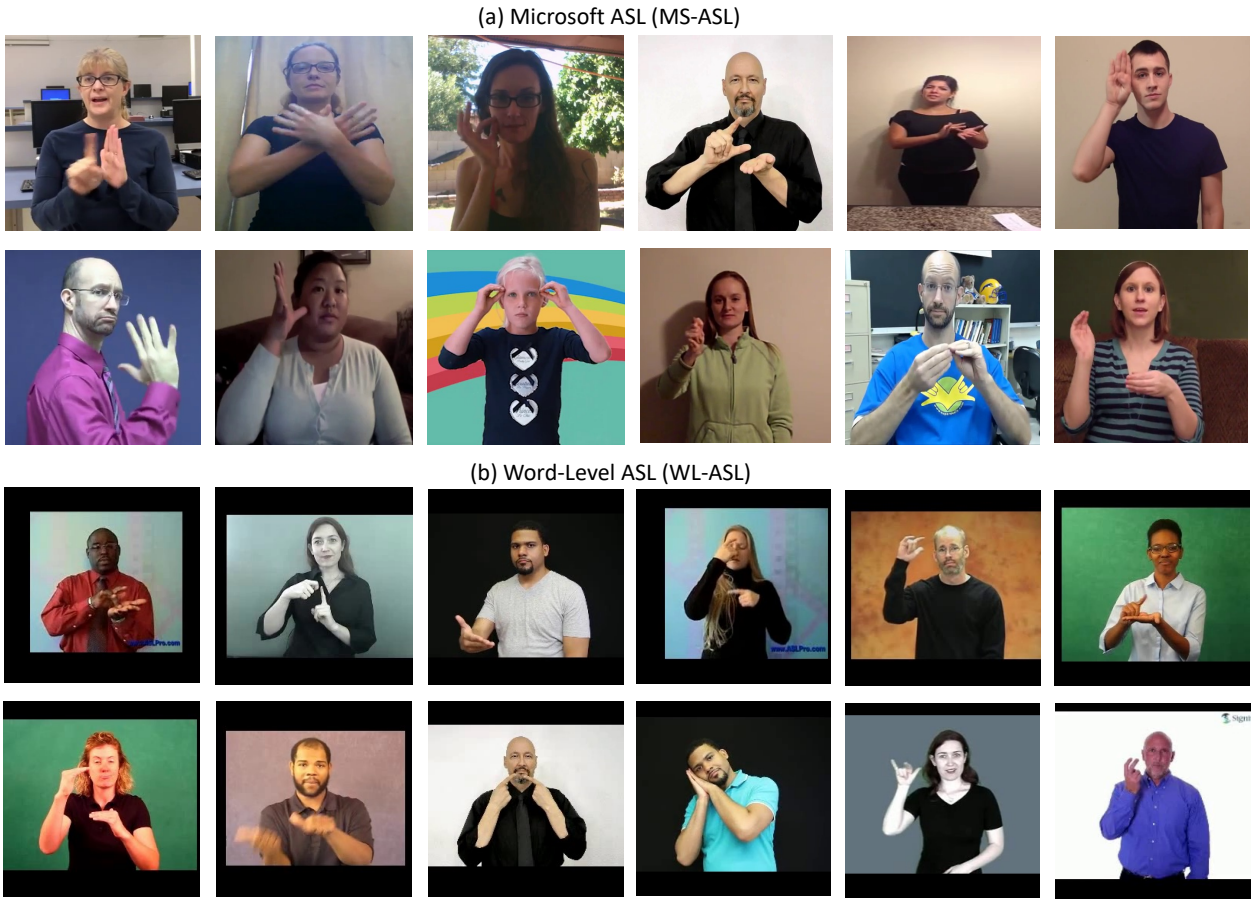


Figure 1.2: Some examples from (a) MS-ASL and (b) WL-ASL datasets. Both datasets have high variety in background colors and appearances, but MS-ASL also includes non-plain backgrounds with furniture and in some cases side view of the signer.

MSASL-200 has the most frequent 200 signs, and so on. Dividing the benchmark this way is useful for studying the effect of the frequency of samples on the test accuracy. In our experiments, we use the smallest subset (MSASL-100) and the complete set (MSASL-1000), since the results of the other subsets fall between them. Table 1.2 shows statistics of these two subsets.

1.3.3 WL-ASL

Word-Level American Sign Language (WL-ASL) [26] is another large ASL benchmark containing 2000 words released in 2020 by the Australian Centre for Robotic Vision (ACRV). The videos in this dataset were collected from 20 different websites including educational sign

Table 1.2: Comparison of AUTSL, MS-ASL, and WL-ASL datasets statistics [26–28]. The row 'Mean' indicates the average number of videos per class.

Dataset	**AUTSL**	**Microsoft (MS-)**		**Word-Level (WL-)**	
Subset	–	**ASL100**	**ASL1000**	**ASL100**	**ASL2000**
Classes	226	100	1000	100	2000
Signers	43	189	222	97	119
Videos	38,336	5,736	25,513	2,038	21,083
Mean	169.6	57.4	25.5	20.4	10.5
Duration [hrs:min]	∼22 hrs	5:33	24:39	–	14 hrs

language websites as well as ASL lessons and tutorials on YouTube [26]. Unlike MS-ASL, WL-ASL is more curated and most of the videos have a plain background and face forward view of the signer (Figure 1.2 b).

The labels (classes) of the video were taken from its titles on YouTube or from the annotations on the educational ASL websites. The temporal boundaries of a sample are the start and end frames of the sign and any repetition of the sign afterward would not be included within these boundries. YOLOv3 [30] and FaceNet [31] were applied by the dataset creators to detect the bounding boxes and the signer identity, respectively. Similar to natural spoken languages, a sign in ASL can be performed in multiple dialects or variants. The annotators of WL-ASL thus made sure every variant of a sign in the test set has examples in the train and validation sets. The gloss labels, temporal boundaries, bounding boxes, signer id, variation id, and other metadata are available as a text file from the dataset repository [29].

After sorting the classes in descending order according to their number of samples, WLASL is divided into four benchmarks (subsets) named WLASL-100, WLASL-300, WLASL-1000, WLASL-2000, where the number of classes in the subset is indicated after 'WLASL'. Similar to MSASL, we select the smallest subset (WLASL-100) and the complete set (WLASL-2000) for our experiments, as the results of the others fall between them. The WLASL dataset follows a ratio of 4:1:1 for splitting the samples between train, validation, and test

sets. Table 1.2 shows the statistics of WLASL and its two subsets in comparison to MSASL and AUTSL. MSASL has more signers, more videos, more samples per class, and more video hours than WLASL. AUTSL has fewer signers but substantially more training videos—in total and per class—than the other two.

1.4 Challenges in Sign Recognition

A notable challenge in automated sign language recognition lies in subtle differences in performing a sign, which can results in different meanings. Each sign is characterized by five parameters that distinguish it from other signs, as outlined below [32].

1. Handshape: The configuration of the hand, including the arrangement of fingers, such as whether they are extended, bent, or folded.

2. Palm Orientation: The direction the palm faces, such as upward, downward, inward, or outward.

3. Movement: The type and direction of motion performed by the hand, such as circular, horizontal, vertical, tapping, or back-and-forth movements.

4. Location: The specific area on the body or within the signing space where the sign is performed, such as near the chin, shoulder, forehead, or other reference points.

5. Non-Manual Signals: Facial expressions, head movements, body posture, or mouthing that convey additional meaning, such as indicating a question, negation, affirmation, or tone. However, these signals often serve as grammatical markers within a sequence of signs rather than being intrinsic to an individual sign.

All five parameters are combined simultaneously in the production of each sign. Altering any one of them can change the meaning (Figure 1.3). For instance, the ASL signs for ‘proof’ and ‘stop’ have similar hand movement but differ in the orientation of the right hand. In

contrast, the ASL signs for 'school' and 'paper' have the same hand orientation but differ in right hand movement.

Figure 1.3: Subtle differences in hand movement and orientation can translate into different meanings. The four ASL signs shown here have generally similar two-hands expressions. However, the right hand in 'school' moves (claps) vertically while in 'paper' it moves (claps) horizontally. The orientation of the right hand is parallel to the left hand for 'proof' and perpendicular for 'stop'.

Another challenge is due to regional dialects, where the same word can be signed differently depending on the geographical area. For example, there are 5 distinct ways to sign 'computer' [27].

Additionally, the datasets used for sign recognition introduce their own challenges. First, many classes in MSASL and WLASL have limited training and testing samples. This is significant because deep neural networks with millions of parameters are "data-hungry",

and the quality and quantity of training data play a critical role in the model's performance. Second, as mentioned in Sections 1.3.1 and 1.3.2, AUTSL and MSASL have many variations in backgrounds, lightening conditions, and field of views. Third, because the labeling and temporal segmentation of videos in MSASL were automated, some of its annotation are inaccurate.

Moreover, a considerable number of the YouTube videos referenced in MSASL and WLASL datasets have been removed by the hosting YouTube channels. However, we were able to obtain copies of the missing videos in WLASL after contacting the authors. We reached out to the authors of MSASL but did not receive a response. Nevertheless, we were able to recover some of the missing clips in MSASL from WLASL. Ultimately, 11.4% of training clips, 14.3% of validation clips, and 3.8% of test clips in MSASL-1000 remains missing.

1.5 Contributions

While 3D convolutional models and optical flow modality have been widely used in state-of-the-art isolated sign recognition, a systematic evaluation of their contribution to learning the motion of signs has been limited. This research fills this gap by examining the improvement of 3D convolutional models (slow fusion) over 2D baseline (late fusion) and analyzing the motion components of optical flow. Our results show that for isolated SLR, the improvement achieved by slow fusion compared to late fusion methods is more pronounced than that for action recognition. Furthermore, the research challenges conventual assumptions about optical flow, showing that its primary value lies in segmenting relevant regions rather than capturing motion magnitude or direction. Remarkably, replacing optical flow with simpler alternatives, such as a binary mask, achieves comparable accuracy.

Skeleton-based sign recognition has been adopted in multiple state-of-the-art methods. However, extracting skeleton data (coordinates of body and hand joints) from RGB input video often relies on a computationally intensive model [33] to achieve a high detection rate.

Instead, we utilize a real-time pose and hand detection module [34, 35], designed for mobile devices, and demonstrate that a graph convolutional network (ST-GCN) is better suited for this modality compared to recurrent networks or transformer encoders. To enhance the interpretability of ST-GCN, we devise a visualization method based on the edge importance weighting learned by the graph.

Transfer learning from the Kinetics dataset has been proven useful in the action and sign recognition literature. We extend this approach by additionally pretraining the model on the AUTSL dataset, before training on the target dataset (e.g., MSASL or WLASL). We demonstrate that this method (coarse-to-fine training) improves the accuracy across various modalities, including RGB, optical flow, and skeleton. Our ablation studies show that for this method to be effective, the datasets used for pretraining should be of high mass and high density.

A key contribution of this work is the systematic evaluation and analysis of stream fusion strategies and the possible combinations of two- and three-stream networks using RGB, optical flow, and skeleton inputs. Our findings indicate that combining RGB and skeleton streams achieves the best performance while adding additional streams leads to diminishing returns. Furthermore, our results demonstrate that the optical flow stream can be replaced by another RGB stream without a meaningful loss in accuracy, whereas adding the skeleton stream yields the most significant gain over the RGB baseline due to greater diversity in predictions.

Our two-stream network [I3D, GCN] achieves new state-of-the-art accuracy on the MSASL and WLASL datasets while consuming significantly fewer FLOPs than other state-of-the-art methods. This is significant as it demonstrates that by focusing on improving the training and fusion strategies—without relying on optical flow or unnecessary architectural complexities—higher sign recognition accuracy can be achieved while maintaining computational efficiency.

1.6 Overview of Chapters

The subsequent chapters in this dissertation are organized as follows.

- **Chapter 2 (Background)** includes a literature review and critical analysis of various spatiotemporal models from action recognition that can be applied to isolated sign language recognition.

- **Chapter 3 (Methods)** includes the design of individual streams (models) and the methods for extracting optical flow and skeleton data from RGB inputs. The implementation details of the training and inference pipelines are also provided. Several stream fusion methods are proposed and tested.

- **Chapter 4 (Results and Analysis)** includes the ablation studies, experimental results, and analysis of the effects of slow fusion, optical flow, skeleton models, and transfer learning. The results of two- and three-stream networks are provided, analyzed, and compared with state-of-the-art approaches.

- **Chapter 5 (Conclusions)** summarizes the research findings and outlines directions for future work.

Chapter 2

Background

Convolutional neural networks are regularized forms of neural networks that use convolution instead of full matrix multiplication in one or more of its layers. Although they have been known since the 1980s [36, 37], practically showing that they are the best paradigm for learning image classification on a large scale is attributed to Krizhevsky, et al. [38]. In the ImageNet challenge of 2012, their work demonstrated that a deep convolutional neural network outperforms SVM classifiers, with state of the art hand-engineered features, by a large margin on ImageNet dataset [39]. The prevailing success of deep learning models in vision applications afterward can be ascribed to three main factors: i) the availability of large-scale annotated datasets (e.g., ImagNet), ii) exploiting the high parallelism of GPUs for training, and iii) development of architectural designs to increase accuracy and training efficiency through iterative trial-and-error.

A similar shift from feature engineering towards training deep learning models on raw data has taken place for videos. Dense trajectory methods [40, 41] which are based on extracting spatial and temporal features (e.g., HOG, HOF, MBH) along motion trajectories in videos were known for a while as the state-of-the-art method on relatively small video datasets, such as UCF-101 and HMDB-51. As larger video benchmarks became available (e.g., Kinetics), deep learning models surpassed linear classifiers that used hand-engineered features, due to the fact that they became less likely to overfit while having more capacity to learn from the data.

Training deep learning models on videos, however, is not a free lunch that comes without difficulties. Compared to image classification, action classification presents additional challenges due to variations in viewpoints and motion. Therefore, it might require more training data than ImageNet for it to work as well [10], which in turn requires more training time and hardware resources. For this reason, we review deep learning models for videos from

both the modeling and the computational perspectives. From a modeling perspective, we are interested in the question of how well spatiotemporal models capture motion information through their different architectures. This can be measured quantitatively by how much they improve over the accuracy of a 2D ConvNet baseline which disregards temporal ordering and dependencies between the frames. From a computational perspective, we are interested in the question of how the model architecture contributes to its computational efficiency as measured by the number of Floating Point Operations (FLOPs). Using the insight given by these perspectives, a trade-off can be made between the accuracy improvement and the computational cost of the model.

The rest of this chapter is organized as follows. Section 2.1 introduces the single-image baseline and describes the three broad categories of fusing temporal information across the frames. In Section 2.2, we summarize the characteristics of four action recognition datasets used to train and test the models in this chapter. In Section 2.3, we review common 3D ConvNet architectures where motion cues are learned implicitly. Section 2.4 describes two-stream networks in which the second stream is claimed to learn motion patterns using optical flow or fast pathway. Section 2.5 describes two ways of combining recurrent units with convolution to capture temporal ordering and long-range dependencies. Non-local networks, which use a general form of attention, are detailed in Section 2.6. We provide our critical assessment (strengths and weaknesses) of the models in Sections 2.3 through 2.6 in their respective Analysis subsection. Section 2.7 summarizes common training and testing methods in the literature. The chapter concludes with a summary of key observations that motivate our research in Section 2.8.

2.1 Fusion Methods

A simple multi-frame baseline to consider for action recognition tasks is to pass the frames in the video individually through a 2D ConvNet and average the individual predictions from the frames to get a single prediction for the video. This way, the frames in the video are

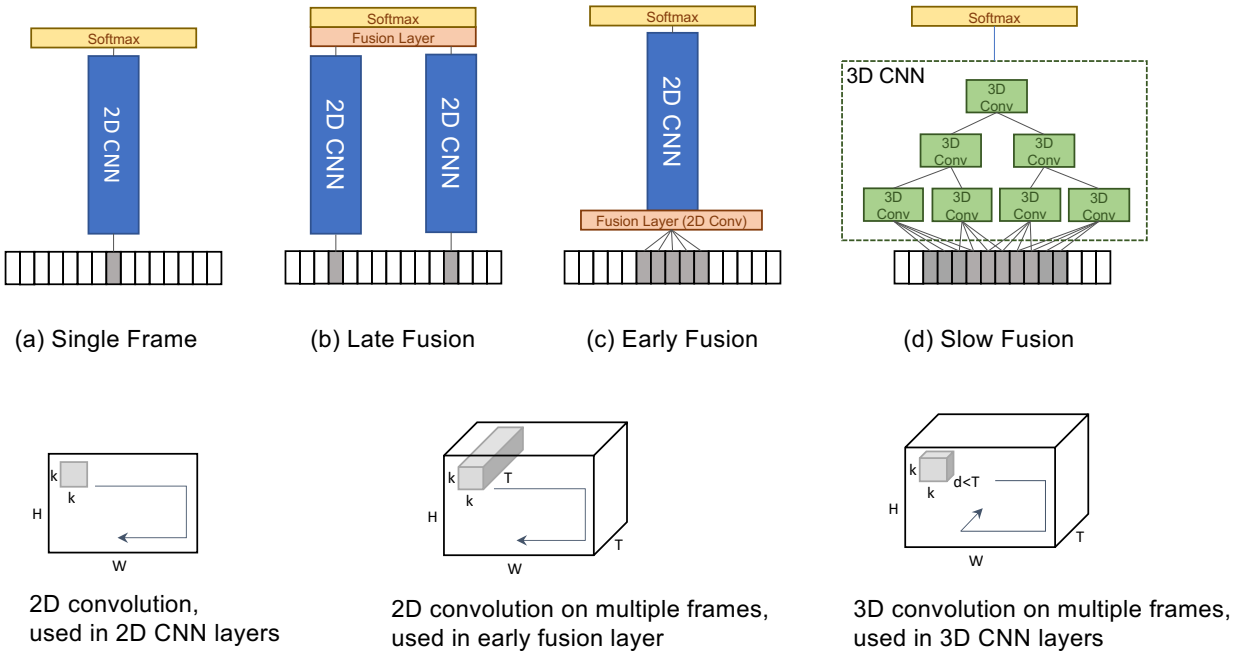


Figure 2.1: Methods for temporal fusion through the network. First row shows the single frame baseline and the three methods commonly used for fusing information from frames in the clip [7]. Second row shows how a 2D or 3D convolutional kernal is applied on single or multiple frames [1].

treated as independent images each voting on the class of the action. Alternatively, a single-frame baseline is commonly used in action recognition benchmarks [7,14,42] where one frame from the video is selected and passed to a 2D ConvNet to get a prediction for the action class in the video (Figure 2.1 a). Since these baselines do not take into account any temporal ordering or relations between the frames, they are useful in understanding the contribution of static appearance alone on the classification accuracy, and by how much the accuracy is improved once the temporal information is considered.

To learn the motion features of actions, however, the temporal domain should be considered by the model. Karpathy, et al., [7] investigated three broad approaches to fusing information across the frames:

Early Fusion. In early fusion, information from multiple frames is integrated early, at the pixel level. This is achieved by concatenating the frames along the channel dimension to form a single input volume with $C \times T$ channels, where C is the number of channels (e.g., 3 for RGB frames) and T is the number of frames. The 2D convolutional kernel in the fusion

layer spans the depth of the input volume and outputs 2D feature maps. Therefore no other changes are introduced in subsequent layers of the 2D ConvNet (Figure 2.1 c). The purpose of this approach is to capture local motion cues early from raw pixels, although it tries to achieve that very quickly using only one layer.

Late Fusion. In contrast to early fusion, late fusion merges information from the frames late in the network, after they have been processed individually by all convolutional layers. Karpathy et al. [7] used two single-frame towers with shared parameters placed 15 frames apart (Figure 2.1 b). The late fusion layer in that instance is a fully-connected layer (fc6) that merges the high-level features to detect global motion across the two frames. This approach can be generalized to multiple frames, and by using other types of fusion layers, such as recurrent layers [11, 43], average pooling [9], or max-pooling [11]. Learning local motion across the frames in late fusion is difficult because the fusion layer receives high-level representations of the frames where subtle local motion is harder to discern.

Slow Fusion (aka 3D ConvNet). A balanced approach between early and late fusion is slow fusion, where 3D convolutions are used in the convolutional layers. While a 2D convolutional kernel in early fusion has a full temporal receptive field T, a 3D convolutional kernel typically has a small temporal depth $d < T$ allowing the ConvNet to slowly integrate motion features from the frames through its convolutional layers (Figure 2.1.d). This is similar to the way 2D ConvNets gradually fuse spatial features throughout the network. In the case of 3D ConvNet, however, slow fusion occurs for both the spatial and motion features. In this approach, both local and global motion can be captured. However, it has the disadvantage of increasing the number of parameters in the model, thus increasing the computations and the size of the dataset required for training without overfitting. In Section 2.3, we describe some instantiations of 3D ConvNet architectures with the slow fusion approach.

2.2 Action Recognition Datasets

In this section, we present an overview of the action recognition datasets related to the results discussed in this chapter. This should add context to the models in terms of the scale and the intricacy of the data they were trained on.

UCF-101 [5]. UCF stands for "University of Central Florida". This widely used dataset for benchmarking action recognition models includes a total of 101 action classes, divided into five types: human-object interaction, body-motion only, human-human interaction, playing musical instruments, and sports. The dataset has 13,320 trimmed videos downloaded from YouTube with 320×240 resolution. The total duration of all videos is 27 hours, with a mean video length of 7.21 seconds.

HMDB-51 [6]. HMDB stands for the "Human Motion DataBase". This dataset has 51 action classes with a total of 6,766 videos extracted mostly from movies, with a small portion from YouTube and other sources. The 51 actions can be grouped into five general categories: general facial actions, facial actions with object manipulation, general body movements, body movements with object interaction, and body movements for human interaction.

Sports-1M [7]. This is a large scale dataset that consists of 1 million Youtube videos with 487 classes of sports. There are 1000-3000 videos per class. This dataset is annotated with many types of sports such as aquatic sports, team sports, winter sports, ball sports, combat sports, sports with animals, etc. Some sports are fine-grained; for instance, it has seven different types of American football.

Kinetics-400 [8]. The state of the art performance on UCF-101 has nearly plateaued after I3D [2], and the Kinetics dataset from DeepMind has become the new benchmark for evaluating the state of the art models. Kinetics-400 has 400 human action classes with 400-1150 videos for each action. In total, it has 306,245 videos obtained from YouTube with each video lasting around 10 seconds. The list of action classes covers: single person actions, person-person actions, and person-object actions. Kinetics-600 and Kinetics-700 are extended versions of this dataset with 600 and 700 action classes, respectively.

All these datasets have a variety of camera motion and viewpoints as well as different lighting conditions, shadows, and backgrounds.

2.3 Network Architectures for 3D CNN

In this section, we describe prominent architectures for 3D convolutional neural networks that demonstrated competitive performance on action recognition datasets. These models follow the slow fusion approach since many of their layers use 3D convolutional filters with a small temporal depth.

2.3.1 AlexNet

AlexNet [38] was the first deep learning model to win the ImageNet classification challenge (ILSVRC-2012). The input image to the network is of size 224×224. The image is processed by five 2D convolutional layers of kernel sizes 11, 5, 3, 3, and 3, with max-pooling after the first, second, and fifth layers. The output from the last pooling is flattened and fed to two consecutive fully connected (dense) layers of size 4096, followed by a softmax which outputs probabilities for the 1000 classes in ImageNet.

To extend the standard AlexNet for slow fusion (i.e. 3D AlexNet), Karpathy et al. [7] added a temporal dimension to the 2D spatial filters in the first three convolutional layers. Filters in the first convolutional layer use a temporal depth of $d = 4$ with stride 2 and are applied to $T = 10$ frames, thus producing 4 responses. For the second and third convolutional layers, a temporal depth of $d = 2$ with stride 2 is used, resulting in the third layer having a full temporal receptive field of 10 frames. Consequently, the fourth and fifth convolutional layers receive inputs of unit temporal length so they are not modified.

A multiresolution AlexNet architecture is proposed in [7] to speed up the training process of the network by using two streams of similar architecture, to process two different spatial resolutions of the frame. The context stream operates on the full frame downsampled to half the original resolution. The fovea stream operates on the full resolution of the frame

cropped from the center to be half the original size. The fovea stream makes use of the observation that objects of interest are more likely to be seen close to the center of the image. Both streams are fused before the first fully connected layer by concatenating their activation maps. As a result, the total number of floating point operations (FLOPs) in the convolutional layers of both streams is half of that in the original architecture for the trade-off that they hold twice the numbers of parameters.

2.3.2 C3D Network

C3D (Convolutional 3D) network [1] is a 3D ConvNet that follows a simple architectural design based on VGG net [44], the winning model for the 2014 ImageNet classification challenge. Unlike AlexNet, the VGG model adopts a homogeneous setting of 3×3 kernels in all its convolutional layers. The smallest instantiation of a VGG net (VGG11) has 11 layers including 8 convolutional layers, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. The channel depth after each downsampling with a pooling layer is doubled to capture more complex features in later layers while keeping a similar amount of computations.

C3D is the 3D version of VGG11 where every convolutional kernel in the network gets a temporal depth of $d > 1$. Tran et al. [1] conducted a kernel temporal depth search and found that using a kernel of size $3 \times 3 \times 3$ in all convolutional layers slightly outperformed other temporal depths such as 1, 5, or 7. A nice attribute of C3D is the simplicity and uniformity of its design. The main drawback, however, is the huge memory footprint and FLOPs required for each pass, as well as the number of parameters which are mostly concentrated in the last fully connected layers.

2.3.3 Inflated 3D ConvNet (I3D)

I3D (Inflated 3D) networks are ConvNets with pretrained 2D kernels inflated to 3D. The term was originally used in reference to the inflated batch-normalized inception network (BN-Inception I3D) [2], but it has since been frequently used to refer to other ConvNets

with inflated pretrained kernels. Other instantiations of I3D use ResNet-50 or ResNet-101 [9,45]. In this subsection, we use the term I3D to refer to the original instantiation with the inception architecture.

BN-Inception [46,47] is a 22-layer 2D ConvNet with multiple identical building blocks known as inception blocks. Inception blocks perform convolutions of different kernel sizes as well as max-pooling in parallel. Compared to VGG, Inception-v1 achieves a higher accuracy on ImageNet while consuming less amount of computations and memory. To attain such efficiency, it utilized useful network architectural designs such as stem downsampling, global average pooling, and 1×1 convolutions to reduce the number of channels within the blocks.

Developing new competitive 3D ConvNet architectures from scratch tailored for videos can be a long iterative process with many trials and errors, which is both time-consuming and resource-intensive. I3D models take a different approach by recycling off-the-shelf state of the art 2D ConvNets and convert them into 3D [2,12]. I3D proposes two main modifications to achieve that:

Inflating 2D kernels into 3D. All 1×1 and 3×3 convolutional kernels as well as 3×3 max-pooling in the inception blocks are inflated temporally to have a size of $1 \times 1 \times 1$, $3 \times 3 \times 3$, and $3 \times 3 \times 3$, respectively. This follows a similar conversion pattern as the C3D network.

Bootstrapping from pre-trained 2D filters. Unlike C3D, which was trained from scratch, I3D inherits its initial set of weights from BN-Inception trained on ImageNet. The 3D kernels in I3D are initialized from the 2D filters by repeating the weights of all spatial filters d times (d=kernel temporal depth) along the time dimension and dividing the kernel weights by d. This way, if all the frames in the video were just copies of one image, the output from the I3D network would be equivalent to its 2D counterpart applied to that image.

2.3.4 Analysis

Strengths

+ Adapting successful 2D ConvNet architectures for 3D gives the models a head start by inheriting design principles that have shown to work well on the image domain. This has the added advantage of transfer learning from the large ImageNet to small video datasets. In particular, I3D shows a useful strategy for transfer learning from 2D to 3D convolutional filters. Initializing I3D from ImageNet consistently boosts its accuracy on UCF-101 and HMDB-51 [2].

+ I3D, with Inception or ResNet as its backbone, is much more computationally and memory efficient than C3D with a VGG backbone. In our measurements[1], I3D-Inception is 5.6x faster than C3D (27.8 vs 154.5 GFLOPs), and holds 6.25x fewer parameters than C3D (12.8M vs 80M). This is mainly due to the efficient design concepts that were introduced after VGG, such as 1×1 convolutional bottlenecks to reduce the channel depth, and average pooling to eliminate fully connected layers, which hold most parameters.

Weaknesses

- 3D ConvNets do not improve over the 2D baseline accuracy by much. For instance, AlexNet with slow fusion has only 0.9% better accuracy on Sports-1M than the single-frame AlexNet [7]. Similarly, inflated ResNet-101 shows 1% better accuracy on Kinetics than its 2D version [9]. This can be attributed to the spatial features in these datasets being more discriminative than motion cues. In other words, the datasets do not contain enough actions with similar spatial semantics but different types of motion.

- 3D ConvNets are more susceptible to overfitting since they have many more parameters than 2D ConvNets due to the added kernel dimension. This can be mitigated by data augmentation or pretraining on large datasets. In fact, the success of I3D over other

[1]We used ptflops [48] to run the measurements

contemporary models can be largely attributed to its pre-training on Kinetics, as observed by the results of I3D [2] on UCF-101 and HMDB-51 datasets.

2.4 Two-Stream Networks

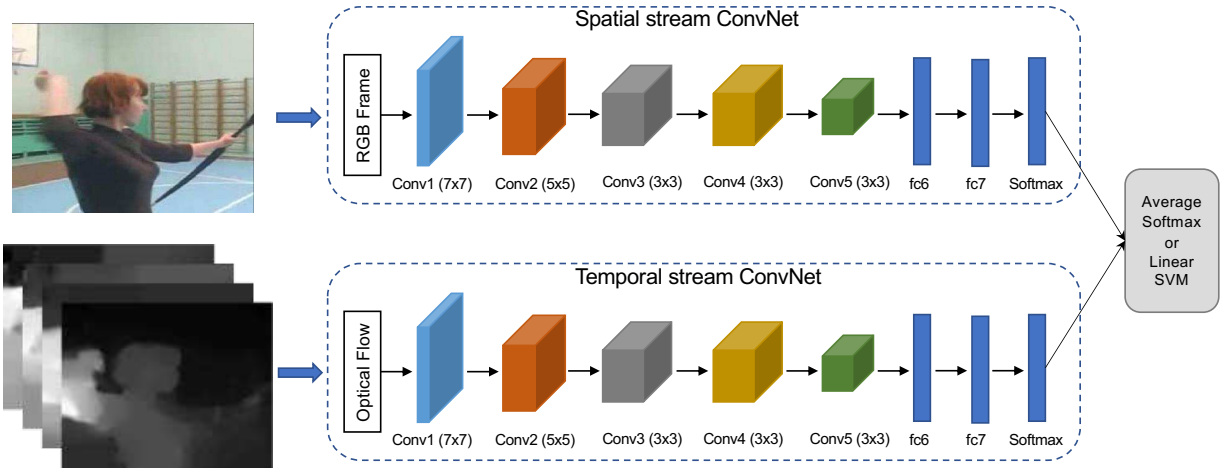


Figure 2.2: Two-stream network architecture for video classification. The spatial stream operates on a single RGB frame while the temporal stream operates on the optical flow computed from multiple consecutive frames [14].

Two-stream networks refer to the general framework of using two network streams; one to model spatial appearance and another to model the motion of the action. In 3D CNNs, learning motion cues is intertwined with learning the appearance features, since they treat the temporal dimension as a third spatial dimension. In contrast, two-stream networks claim to learn motion patterns explicitly through a dedicated temporal stream [14, 49] in addition to spatial stream.

The first non-shallow two-stream neural network used for action recognition was proposed by Simonyan et al. [14]. In their seminal work, both streams have similar 2D CNN architectures to AlexNet [38] (Figure 2.2). The spatial stream operates on a single frame from each video chosen randomly akin to the single frame baseline in [7], which showed a competitive performance to slow fusion. The spatial stream can contribute to the action recognition task because some actions are associated with the existence of a particular object in a still frame.

Since the spatial stream is essentially an image classification architecture, it can also benefit from pre-training on ImageNet. The second stream, referred to as the temporal stream, uses the optical flow of the video for its input modality instead of RGB frames.

2.4.1 Optical Flow

To explicitly represent motion between video frames, the temporal stream CNN operates on optical flow displacement fields between consecutive frames. Optical flow is the apparent motion of individual pixels from one frame to the next [50]. Optical flow estimation is often based on the two assumptions of brightness constancy and smoothness of motion [51]. Several analytical methods were devised for optical flow estimation [52–54], each with their assumptions and shortcomings. However, deep learning methods for optical flow estimation [55, 56] have become more prominent as they achieve state-of-the-art accuracy. Given that the optical flow field has been computed between each pair of consecutive frames in a video, Simonyan et al. [14] proposed the following three ways to stack the displacement vector fields to form the input for the temporal stream.

Optical flow stacking. An optical flow displacement field is a set of vectors representing how points have moved from one frame to the next. For a point (u, v), $\mathbf{d}_t(u, v)$ denotes the displacement at that point from frame t to frame $t + 1$. The displacement vector field has horizontal and vertical components (d_t^x, d_t^y), thus can be represented by two channels. By stacking the optical flow of L consecutive frames, the input volume will have $2L$ channels. This can be arranged such that vertical and horizontal channels are placed in the even and odd indices, respectively.

Trajectory stacking. Instead of sampling points at the same locations across multiple frames, the motion trajectories of the sampled points can be used as an alternative representation. To find the trajectory of a point in an initial frame, the displacement vector at that point is applied to get its location in the second frame. The displacement vector at the new location in the second frame is then applied to get its next location in the third frame, and

so on. This method was inspired by dense trajectories [40], which showed a state of the art performance using hand-crafted features. Trajectory stacking has the advantage of explicitly encoding the motion trajectories in the input volume. However, the errors in optical flow estimation can add up, causing deviations from true trajectories.

Bi-directional optical flow. The previous two representations use forward optical flow where displacement vectors follow the natural direction of time. Backward optical flow estimates reverse motion from a frame t to the previous frame $t-1$. To form a bi-directional optical flow, the input volume, representing L consecutive frames, is divided into two halves where each $L/2$ represents one direction of optical flow.

The temporal and spatial streams can be trained jointly or separately and the softmax outputs from each stream can be averaged or combined and passed into a linear SVM classifier.

2.4.2 Analysis

Strengths

+ In Simonyan et al. [14], the temporal stream with optical flow stacking achieves 8.3% improvement in accuracy on UCF-101 over the spatial stream. This margin becomes even more significant when the spatial stream is trained from scratch. Specifically, the temporal stream shows 24.6% better accuracy than a slow fusion CNN operating on a stack of 11 RGB frames without any pretraining. This indicates that optical flow as a feature extractor is particularly useful in situations where pretraining or access to more training data is not feasible.

+ The study in [14] demonstrates that multitask learning for the temporal stream and transfer learning for the spatial stream are valuable tools to reduce overfitting, especially on small video datasets. Training the temporal stream on HMDB-51 and UCF-101 in a multitask fashion (i.e. one backbone with two softmax classifiers, one for each dataset) achieves 8.8% improvement in accuracy on HMDB-51 over training on HMDB-51 without

any additional data. Moreover, pretraining the spatial stream on ImageNet adds 20.7% to its accuracy on UCF-101 relative to training it from scratch.

Weaknesses

- Although the temporal stream outperforms the spatial stream in [14], no conclusive evidence was provided to demonstrate that the temporal stream with optical flow input is better at learning motion patterns than a 3D convolutional network with RGB input. Carreira et al. [2] compare a two-stream network of RGB and optical flow inputs to a single stream RGB baseline, but no comparison is made to a two-stream network with RGB input only. This is needed to discern whether the value added by the additional stream is due to optical flow or simply the ensemble of models.

- Estimating optical flow is based on the assumptions of constancy and smoothness of motion and thus, the frames have to be consecutive to avoid poor estimations. This involves more computations and memory since reducing the spatial or temporal resolution (via sampling) negatively affects the accuracy of the flow.

2.4.3 SlowFast Network

The SlowFast network, proposed by Feichtenhofer et al. [49], is a two-pathway architecture that operates at two different frame rates. This is analogous to the two-stream architecture [14], which has been the basis for many competitive models in the literature [2, 12, 57]. Unlike the original two-stream network, which used both RGB and optical flow (two different modalities) as inputs, SlowFast networks sample the input frames at two different temporal speeds using a single RGB modality. The SlowFast architecture is set to learn from raw video data end-to-end without using extra hand-designed representations such as optical flow.

The SlowFast network architecture is composed of two pathways, slow and fast. The slow pathway operates at a low frame rate and is meant to capture spatial semantics (e.g., object recognition). The fast pathway is intended to capture motion information at fine temporal resolution by operating on a high frame rate (Figure 2.3). This is motivated by

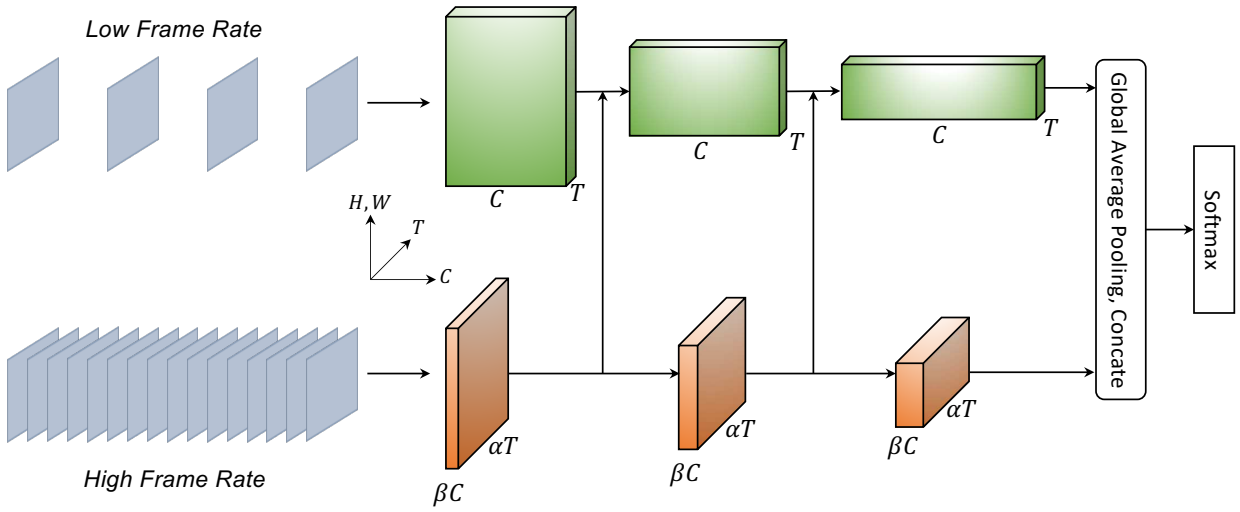


Figure 2.3: A SlowFast network. The slow pathway (green) operates on a few number of frames (T) but has a high channel capacity (C). The fast pathway (orange) has higher temporal resolution (αT) but lower channel capacity (βC) than the slow pathway (where $\alpha > 1$ and $\beta < 1$) [49].

the observation that the categorical semantics of a visual scene in a video such as a person or an object (along with their features, texture, color, etc) change slowly relative to their motion [49]. For action recognition, an entity such as a human or animal can be identified using one or few frames while their actions such as waving, jumping, walking, and running might require higher temporal resolution (i.e. more frames) to be distinguished.

Another inspiration for the two pathway design was taken from the visual system in primates, where Parvocellular neurons (P-cells) are more sensitive to spatial detail and color than Magnocellular neurons (M-cells). M-cells have a poorer spatial resolution but greater temporal resolution than P-cells and thus, are more responsive to fast changes in time [58]. However, this connection to biology is "rough and premature" as admitted by the author [49].

A major difference between the two pathways is their channel capacities. The fast pathway is designed to be very lightweight in terms of computations (FLOPs) and thus, processes significantly fewer channels throughout the network than the slow pathway. Feichtenhofer et al. [49] reported that only 20% of the total computations are carried out by the fast pathway even though it operates on a higher temporal resolution than the slow counterpart. This is because the number of FLOPs performed by a convolutional layer is roughly quadratic

in terms of the channel depth (considering both input and output channels), but linear in terms of its temporal depth.

More concretely, let $\alpha > 1$ represents the frame rate ratio between the fast and slow pathways, and let $\beta < 1$ represents the ratio of channel depth between the fast and slow pathways. Then, the fast pathway samples αT frames and computes βC channels, where T and C are the numbers of frames and channels used in the slow pathway, respectively (Figure 2.3). The experiments in [49] has α set to 8 and β to 1/8. In other words, the fast pathway has 8 times the temporal resolution of the slow pathway but 1/8 of its channel capacity.

Similar to the two-stream networks [14], the SlowFast network concept is generic and can be instantiated with various backbones. In [49] each pathway is a 3D ResNet modified from [12]. Lateral connections are used to fuse the information from the fast to the slow pathway after the first pooling layer, and the second, third, and fourth residual blocks. To match the temporal dimensions before fusing, multiple methods have been explored, including packing all α frames into the channels of a single frame, sampling one frame out of every α frames, or using time-strided convolution. Finally, at the output of each pathway, a global average pooling is applied to get a feature vector. The two vectors are then concatenated and fed into a softmax layer for classification.

2.4.4 Analysis

Strengths

+ The SlowFast model shows better accuracy while being more computationally efficient than the two-stream I3D. Moreover, this is one of the first studies that does not overlook the accuracy vs computational cost tradeoff when comparing the models. For instance, SlowFast ResNet50 (8×8) achieves an accuracy of 77.0% on Kinetics-400, a small margin from 77.7% achieved by Nonlocal ResNet101 [49]. However, the former requires 5.46x fewer

FLOPs than the latter, which translates to faster inference and a significant reduction in computational cost.

\+ The SlowFast results indicate that optical flow is dispensable and not necessary for competitive performance on large video datasets. While using optical flow has been the foundation of several states of the art results on UCF-101, it is a hand-designed representation that adds extra computational overhead and storage cost.

Weaknesses

\- The study interprets lowering the channel capacity for the fast pathway as weakening its ability for building spatial representations. While that is true, it also weakens its ability for capturing motion since all the channels are spatiotemporal. Moreover, the fast pathway contributes only up to a 3% improvement to the accuracy of the slow-only model [49]. Therefore, the fast pathway can not be claimed to capture motion patterns much better than the slow pathway.

\- Ablation studies were provided in [49] for values of β, types of lateral connections, and ways to weaken the spatial input to the fast pathway. However, there is no ablation for varying values of α. This is needed in order to justify the intuition behind the two pathway design. If reducing α (e.g., by lowering the frame rate of the fast pathway) while increasing β has no significant effect on the accuracy, it could imply that the success of the model is not necessarily due to the characteristics of the two pathways, but to the more-than-one-stream architecture. In some way, this resembles the effect of an ensemble of two or more neural networks on the total accuracy.

2.5 Recurrent Convolutional Networks

Recurrent neural networks (RNNs) are a common approach for learning sequence modeling when there is temporal dependence between input tokens as is the case in text, speech, and time-series data. Unlike feedforward neural networks, RNNs use an internal memory

(hidden state) to summarize the sequence in the input it has seen so far. Action recognition from videos seems like a natural fit for RNNs since a video is a sequence of frames with temporal order and structure. However, standard RNNs architectures, such as LSTM or GRU, operate on one-dimensional inputs. For that reason, CNN is usually applied first to encode the frames as a sequence of feature vectors before passing them into an RNN. Recurrent Convolutional Network (RCN) refers to the use of both convolution and recurrent layers in the network. In this section, we describe two methods for combining recurrent networks with convolution.

2.5.1 2D CNN + RNN

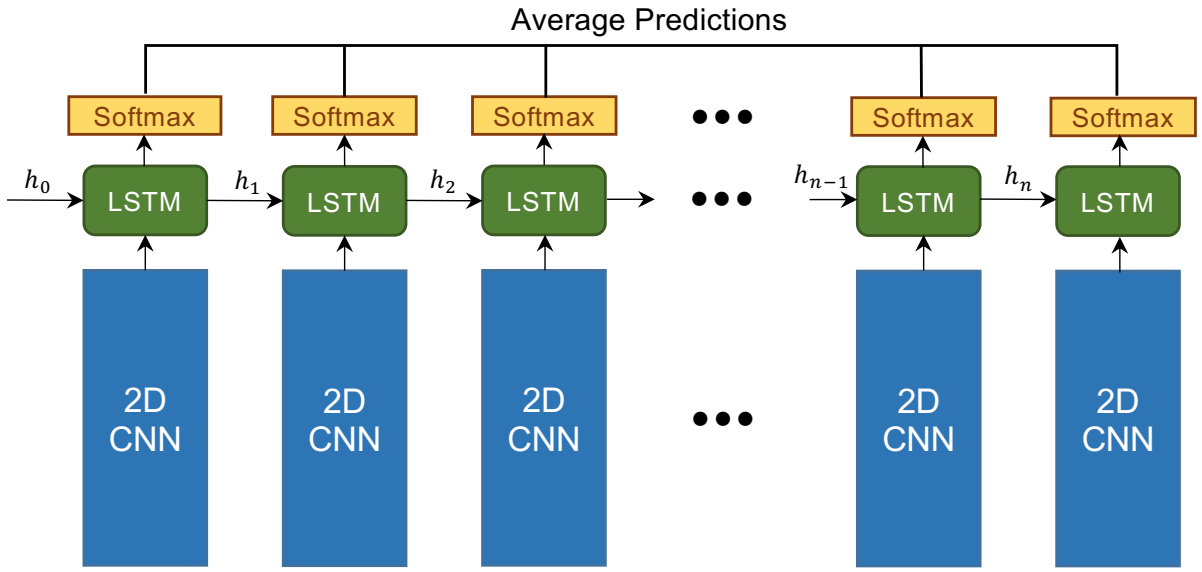


Figure 2.4: Recurrent convolutional network. The 2D ConvNet extract high level features from the frames. The LSTM processes the features of the frames sequentially.

A typical approach for using RNN with videos consists of two steps. First, the frames are converted from 2D images to one-dimensional high-level feature vectors, using a 2D convolutional network. The output feature vectors are then fed sequentially as inputs to an RNN for video classification (Figure 2.4). This method follows the late fusion approach

discussed in section 2.1. Baccouche et al. [59] first proposed this approach for human action recognition, where it was applied on KTH, a small dataset of six actions.

Ng et al. [11] applied this architecture on bigger action recognition datasets [5, 7] of 2-minutes-long videos instead of short snippets to show that a combination of CNN and LSTM is useful for learning global descriptions of the video's long-term temporal evolution.

Long Short-Term Memory (LSTM) is a recurrent neural network first proposed in [60] to overcome the vanishing gradient problem in traditional RNNs, which struggled to learn long sequences. LSTM provides a solution by adding a memory cell and three gates to explicitly have the recurrent unit learn when to forget or update its hidden state with new information. Similar to convolutional layers, multiple LSTM units can be stacked on top of each other to add more depth and levels of abstraction. For instance, Ng et al. [11] used five layers of stacked LSTMs on top of pre-trained AlexNet. The softmax layer is connected to the top LSTM layer instead of AlexNet. To classify a video, the softmax outputs across all frames are averaged to get final probabilities of the action classes [43].

2.5.2 Convolutional GRU

Using high-level visual features as inputs to the RNN has the disadvantage of discarding local motion information since these features, while being highly discriminative, have low spatial resolution [61]. Moreover, top layer representations in the CNN are more translation-invariant since they have passed through multiple max-pooling layers. The previous approach tends to focus more on global appearance changes which is not always the case for actions in videos where frame-to-frame variation is likely to be local and smooth. To capture fine motion patterns, intermediate activation maps from the CNN might also have to be considered as inputs to the RNN. For that purpose, Ballas et al. [61] proposed GRU-RCN, a novel RCN architecture to address the weakness of the late fusion CNN+RNN approach. For clarity and convenience, we refer to the GRU-RCN model as convolutional GRU (or ConvGRU for short) as opposed to the standard fully-connected GRU.

ConvGRU uses convolutional activation maps (referred to as percepts in [61]) from all layers of a ConvNet (Figure 2.5). These maps represent different levels of feature abstractions and spatial resolutions. However, applying a recurrent unit directly on these visual percepts would result in a high number of input-to-hidden parameters due to the large convolutional map size when flattened. ConvGRU solves this by using convolutions instead of full products.

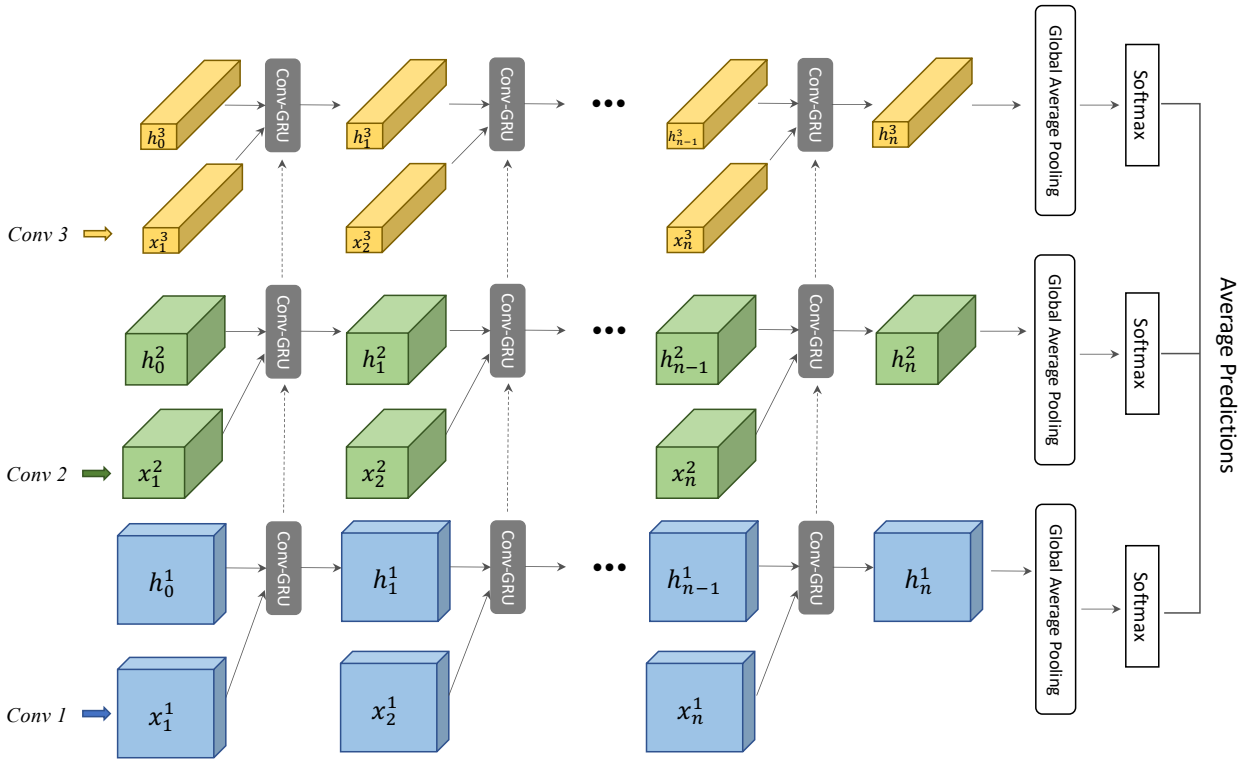


Figure 2.5: Feature maps from different layers of a 2D ConvNet are extracted from the frames to be processed sequentially by ConvGRUs. A different frame is processed at every time step (left to right). x_t^l represents the feature map for frame t extracted from layer l in the ConvNet. Bottom-up connections can be optionally added between ConvGRUs to form a Stacked ConvGRU [61].

The ConvGRU model uses Gated Recurrent Units (GRU) [62] as they have demonstrated similar performance to LSTMs while requiring less memory [63]. The main concept behind the ConvGRU model is to replace the fully connected linear products (i.e. matrix-vector multiplication) in a standard GRU with convolution operations. This helps reduce the number of parameters and enforces sparse connectivity as well as parameter sharing across

different spatial locations and time steps. All gates, inputs, and hidden states in this model are 3D tensors (Figure 2.5). To preserve the spatial dimensions of the hidden states, zero-padded 3×3 convolutions are applied at every time step. Using a small convolutional kernel of size $k \times k$ instead of fully connecting to the entire spatial grid of size $N \times N$, reduces the memory and computations by a factor of $\frac{k^2}{N^2}$.

The ConvGRU model in [61] is applied on five feature maps extracted from the second, third, fourth, fifth pooling layers, and the penultimate fully connected layer (fc7) of pre-trained VGG16 [44]. After the last time step, an average pooling is applied to the hidden representations to reduce their spatial dimension to 1×1. The five feature vectors are then passed into softmax classifiers (one classifier for each vector). The probabilities from these classifiers are averaged to get the final prediction (Figure 2.5).

In the typical ConvGRU model, five units are independently applied to the convolutional maps. Ballas et al. [61] have also investigated two other variants; Stacked ConvGRU and bi-directional ConvGRU. Stacked ConvGRU adds an extra bottom-up connection across the 5 units, thus preconditioning each unit on the output of the one underneath at the current time step. Bi-directional ConvGRU considers both the forward and reverse temporal orders of the frames. In this case, the ConvGRU is run twice, in both directions, and the two final hidden representations are concatenated before passing them into the classifiers.

2.5.3 Analysis

Strengths

+ The ConvGRU model proposes a new layer that performs convolution and recurrent operations at each time step and thus has the advantage of both shift-invariance and internal state in one layer. Moreover, a convolutional GRU is more computationally and memory efficient than a fully-connected GRU for spatiotemporal inputs.

+ Recurrent convolutional networks makes it possible to process arbitrary lengths of frames in a video. A common practice for inference in 3D ConvNets is to trim the video into

short clips of fixed size, then run the model on each clip and average the predictions [2]. RCNs eliminate the need for such practice since they do not use a kernel of fixed temporal length like 3D convolution.

Weaknesses

- Although ConvGRU presents an elegant way of performing recurrent convolution, it is not as competitive in practice. Its accuracy on the UCF-101 dataset is on par with the VGG-16 baseline in [42] while consuming more memory and computations due to its recurrent operations. Moreover, temporal ordering does not seem to be crucial for action recognition datasets. For instance, using temporal max pooling, which is order invariant, to temporally aggregate feature maps from the last convolutional layer yields an accuracy within 1% of CNN+LSTM on UCF-101 and Sports-1M datasets [11].

- Training deep learning models often exploits the high parallelism of GPUs. However, recurrent models are not suitable for parallel computations especially for long sequences due to the recurrences at each layer which are inherently sequential. This drawback is one of the motivations behind pursuing architectures without any sequential operations such as 1-dimensional convolutions [64] or transformers [65].

2.6 Neural Networks with Visual Attention

Attention mechanism in neural networks has emerged as a valuable tool for language modeling, making attention-based architecture (e.g., transformer) as the new de-facto standard for language processing tasks [65–67]. Attention mechanisms made their way first into RNN encoder-decoder architectures for neural machine translation to overcome the limitations of the fixed-length vector representation of a sentence given by the encoder's last hidden state [68], where information from time steps far away might not be well encoded. By introducing attention to this model, the decoder outputs depend on a weighted combination of all hidden states from all time steps in the encoder. Incorporating attention also

adds the advantage of interpretability to the model by visualizing the attention weights on the inputs.

Although attention in neural networks was loosely inspired by the visual attention mechanisms in the human eyes [69,70], its application in computer vision has not been as prominent as in NLP due to the high effectiveness of convolutional neural networks for vision tasks. Wang et al. [9] proposed non-local neural networks as a general approach where attention mechanisms can be applied to visual data such as images and videos. While recurrent and convolutional operations process a single local region at a time, non-local operations compute the output as a weighted sum of features at all positions in the input. This allows the model to directly capture long-range dependencies between features in non-local neighborhoods. In this section, we describe the formulation of non-local operations and non-local blocks as introduced by Wang et al. [9].

2.6.1 Non-local Operation

The non-local operation in [9] was inspired by the non-local means algorithm used for image denoising [71]. A generic non-local operation is defined as:

$$\mathbf{y}_i = \frac{1}{\mathcal{C}(\mathbf{x})} \sum_{\forall j} f(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_j). \tag{2.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output signals of the same size, respectively. i is the index whose response is to be computed. j is the index that enumerates all positions in spacetime. f is a pairwise function that returns a scalar representing a similarity measure of two vectors. g is a unary function that returns a representation of the input at index j. $\mathcal{C}(\mathbf{x})$ is the normalization factor, usually set to be $\sum_{\forall j} f(\mathbf{x}_i, \mathbf{x}_j)$.

The non-local operation distinguishes itself from a convolutional operation by considering all positions in the inputs instead of a small region defined by a kernel size. While a fully-connected layer does take into account all positions in an input, it is essentially different from the non-local operation because the relationship between different positions in the

input $f(\mathbf{x}_i, \mathbf{x}_j)$ in a fully connected layer is not a function of the input data [9], but rather a learned weight W of constant value.

Multiple instantiations were considered for f including:

$$f_{\text{gaussian}}(\mathbf{x}_i, \mathbf{x}_j) = e^{\mathbf{x}_i^T \mathbf{x}_j}, \tag{2.2}$$

$$f_{\text{embedded-gaussian}}(\mathbf{x}_i, \mathbf{x}_j) = e^{\theta(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}, \tag{2.3}$$

$$f_{\text{dot-product}}(\mathbf{x}_i, \mathbf{x}_j) = \theta(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \tag{2.4}$$

$$f_{\text{concatenation}}(\mathbf{x}_i, \mathbf{x}_j) = \text{ReLU}(\mathbf{w}_f^T [\theta(\mathbf{x}_i),\ \phi(\mathbf{x}_j)]), \tag{2.5}$$

where $\theta(\mathbf{x}_i) = W_\theta \mathbf{x}_i$ and $\phi(\mathbf{x}_j) = W_\phi \mathbf{x}_j$ are set as embedding functions with W_θ and W_ϕ being $1 \times 1 \times 1$ convolutional kernels. Likewise, g can be set as an embedding function: $g(\mathbf{x}_j) = W_g \mathbf{x}_j$, where W_g is a $1 \times 1 \times 1$ convolutional kernel, similar to W_θ and W_ϕ.

Using the embedded Gaussian instantiation of $f(\mathbf{x}_i, \mathbf{x}_j)$ makes Eq.(2.1) equivalent to the self attention layer used in the transformer architecture [65]. In that sense, $\theta(\mathbf{x}_i)$, $\phi(\mathbf{x}_j)$, and $g(\mathbf{x}_j)$ compute the query, key, and value, respectively. The expression $\frac{1}{\mathcal{C}(\mathbf{x})} f_{\text{embedded-gaussian}}(\mathbf{x}_i, \mathbf{x}_j)$ computes a softmax function along the j dimension, similar to what is used in the self attention formula [65]. Therefore, self attention operation is a special case of non-local operations when $f(\mathbf{x}_i, \mathbf{x}_j)$ takes the embedded Gaussian form [9].

Wang et al. argued that using the softmax attentional behavior in action recognition datasets was not essential and what mattered was the general non-local behavior provided by f. To show that, the dot product and the concatenation instantiations of $f(\mathbf{x}_i, \mathbf{x}_j)$ were used to demonstrate close results to the embedded Gaussian version.

2.6.2 Non-local Block

Using the non-local operation defined in Eq.(2.1), a non-local block (Figure 2.6) that can be inserted into any ConvNet architecture is defined as:

$$\mathbf{z}_i = W_z \mathbf{y}_i + \mathbf{x}_i \tag{2.6}$$

where $\mathbf{x}_i$ and $\mathbf{y}_i$ are the input and output signals, respectively, as defined in Eq.(2.1). The summation in Eq.(2.6) acts as a residual connection so that $\mathbf{z}_i = \mathbf{x}_i$ when W_z is initialized to zero. As such, a pretrained model with one or more non-local blocks starts from the same initial behavior during training as before inserting the blocks.

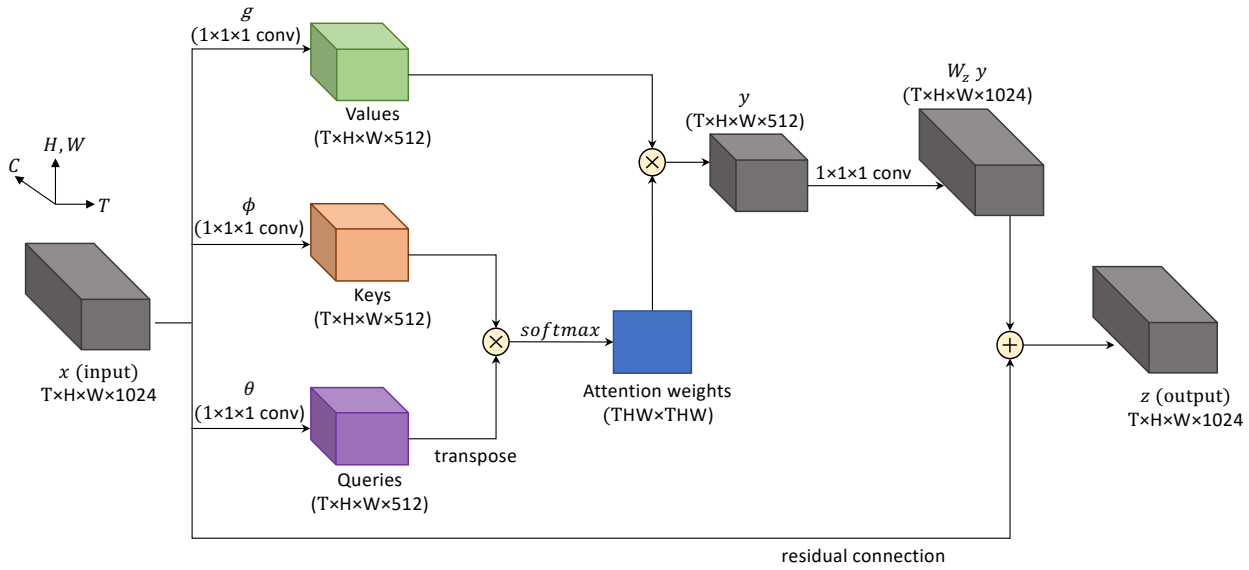


Figure 2.6: Non-local block [9] with the embedded Gaussian version closely resembles a self attention layer. The linear embedding functions θ, ϕ, and g are used to compute the queries, keys, and values from the input, respectively, as well as to reduce the number of channels from 1024 to 512 using $1\times1\times1$ convolutions. $\bigoplus$ denotes element-wise sum and $\bigotimes$ denotes matrix multiplication.

Non-local operations are computationally intensive due to the pairwise computations, especially for high dimensional inputs such as videos. The following three ways were proposed in [9] to reduce the computations in non-local blocks.

High-level insertion. Inserting the non-local block in high-level layers where the feature maps dimensions (H, W, and T) are relatively small.

Bottleneck design. Using W_θ, W_ϕ, and W_g to reduce the number of channels by half. W_z which is $1 \times 1 \times 1$ convolutional kernel can then be used to double the number of channels after computing $\mathbf{y}_i$ to match the number of channels in the input ($\mathbf{x}_i$) feature map (Figure 2.6).

Subsampling. Performing spatial max-pooling operation after ϕ and g to make the computation sparser without negating the non-local behavior.

Using the 2D and 3D Resnet architectures [12, 45], Wang et al. [9] analyzed the effect of inserting 1, 5, and 10 non-local blocks on the performance. These were inserted in late intermediate residual blocks such as the third and fourth residual blocks.

2.6.3 Analysis

Strengths

+ The ablation studies elegantly show the consistent positive effect of inserting the non-local blocks on the accuracy of 2D and 3D models. For instance, inserting 5 non-local blocks into a baseline 2D ResNet-101 added a 2% improvement to the accuracy. In contrast, inflating one kernel to $3 \times 3 \times 3$ for every 2 residual blocks added 1% to the baseline accuracy while consuming 50% more FLOPs and 25% more parameters than the model with non-local blocks [9].

+ Using non-local operations with the embedded Gaussian form for f adds interpretability to the model as we can visualize the locations in the input with the highest softmax weights. In the case of videos, this could help debug and understand the influence of other time steps on a certain time step.

Weaknesses

- Unlike convolutional layers, which process one local neighborhood at a time, non-local operations consider all positions in the input. As a result, the computational complexity of a non-local operation is quadratic in terms of its input length, similar to a self-attention layer in transformers [65]. A more practical and computationally efficient solution is to attend to patches of pixels instead of individual pixels as applied in the Vision Transformer (ViT) [72].

- Unlike recurrent layers, non-local operations do not encode the temporal ordering of the input. To address this, positional encoding [65] can be incorporated when the temporal ordering of frames is important, such as in complex gestures or sign language recognition.

2.7 Related Implementations

Besides the model architectures, training and testing methodology can affect the performance and results significantly. In this section, we describe common aspects of training and testing deep learning models on video data.

2.7.1 Training

At each iteration, a batch of videos is sampled from the training set. Then a short clip is sampled, both temporally and spatially, from each video in the batch. We outline the relevant aspects of this training process below.

Temporal sampling is implemented by randomly selecting a segment of 10 frames [7,61], or 64 frames and dropping every other frame [9, 49]. This sampled segment from a video is usually referred to as a clip.

Spatial sampling is performed first by rescaling the clip spatially so that the shorter side has 256 pixels [2, 14] or a random value in the range of [256, 320] pixels [9, 49], while keeping the original aspect ratio the same. Then, a sub-image of size 224×224 pixels is randomly cropped at the same position from all the frames in the clip. The final input to the model is a clip of size $224 \times 224 \times T$ where T is the number of frames in the clip.

Data augmentation is a preventative measure against overfitting [38]. Spatial and temporal sampling are already forms of data augmentation. Additional methods such as random horizontal flipping and RGB jittering are usually used alongside spatial/temporal sampling to further reduce the effects of overfitting.

Mini-batch stochastic gradient descent (SGD), with momentum set to 0.9, is a widely used algorithm for optimizing the weights of neural networks. We notice that optimization methods with adaptive learning rates such as Adam [73] have not been fully

exploited in action recognition tasks. We suggest examining whether using Adam instead of SGD for such tasks provides an advantage in terms of speed and/or accuracy.

The learning rate is a hyperparameter that is initially set based on the validation set. This value is decayed during training by some factor either according to a certain schedule [9, 14, 49], or whenever the validation error saturates [2, 7]. The training is stopped after a preset number of epochs is reached.

2.7.2 Testing (Inference)

To predict the action class of unseen video in the testing set, a common practice is to temporally sample 10 evenly spaced clips from the whole video. These clips are spatially rescaled so that their shorter side is 256 pixels. Then, a fully-convolutional inference is performed on each clip and the softmax scores from all clips are averaged to get a video-level prediction [2].

Alternatively, five sub-volumes of size $224 \times 224 \times T$ are taken from the four corners and the center along with their horizontal flips to obtain 10 inputs per clip. In this case, the final video prediction is obtained by averaging the softmax scores across the clips and their 10 sub-volumes [61].

2.8 Conclusion

In this literature review, we summarized, compared, and evaluated different deep learning paradigms for action recognition in videos, including 2D and 3D ConvNets, recurrent convolutional networks, non-local operations that generalize attention mechanisms for videos, and two-stream networks with one or two modalities. Our analysis considers the trade-off between accuracy improvement over the 2D baseline and the additional computational cost. Notably, 3D convolutional models showed limited gains over their 2D counterparts [7,9], and 2D convolutional models with late fusion via RNN offered insignificant improvement over pooling [11]. A possible reason for these modest gains is that action recognition datasets [5–8] often have

actions that can be inferred from spatial cues, such as objects or background, which provide a stronger signal than motion patterns. In subsequent chapters, we examine the impact of slow fusion compared to late fusion on isolated sign language recognition (SLR) datasets. Isolated SLR can be viewed as a subcategory of action recognition with more sensitivity to motion patterns, thus providing a more rigorous testbed for spatiotemporal models.

Besides model architectures, two approaches have notably resulted in significant improvements: incorporating additional stream and transfer learning. Optical flow has been utilized in two-stream networks [2,12–14] where one stream captures "appearance" features while the other is claimed to capture "motion" features. Although this construction has resulted in significant improvement over a single stream baseline, it is not evident that motion patterns are better learned due to the optical flow input rather than the slow fusion aspect of 3D ConvNets. As seen in the SlowFast network [49], improvements can also be acheived with two pathways and a single modality (RGB). Therefore, the improvements in both approaches were likely due to the ensemble effect, regardless of input modality. Systematic evaluations of two- and three-stream combinations using RGB, optical flow, and skeleton data for sign language recognition should provide better insights into the best combination. Since transfer learning has been highly effective in improving accuracy for action recognition [2, 14], this technique will be further developed and analyzed in later chapters using transfer learning from other action and isolated sign recognition datasets.

Chapter 3

Methods

This chapter is organized as follows. Section 3.1 presents an overview of the multi-stream framework and outlines the plan to address the research questions. Sections 3.2, 3.3, and 3.4 provide the design details of the individual models in this framework, including those with RGB, optical flow, and skeleton inputs, respectively. The implementation details for the training and inference pipelines, along with strategies for acceleration them, are covered in Sections 3.5, 3.6, 3.7, and 3.8. In Section 3.9, several stream fusion methods are proposed and evaluated.

3.1 Multi-Stream Framework

The purpose of the multi-stream framework is to establish a conceptual structure for evaluating and comparing all possible combinations of streams using the same or different input modalities. We define the following properties for the multi-stream framework.

- **Single input source**. RGB is the only input source for all the streams in this framework, from which optical flow and skeleton data are extracted (Figure 3.1). This framework does not rely on additional hardware, such as a second RGB or depth camera.

- **Multi-modal streams**. We systematically evaluate distinct models trained on three input modalities: RGB, optical flow, or skeleton. We refer to models trained directly on the RGB input as raw-RGB models, or simply RGB models. Models trained on the optical flow are referred to as optical flow models, and those on skeleton data as skeleton models. We use the term "stream" interchangeably with "model" to refer to a model that is part of a multi-stream network.

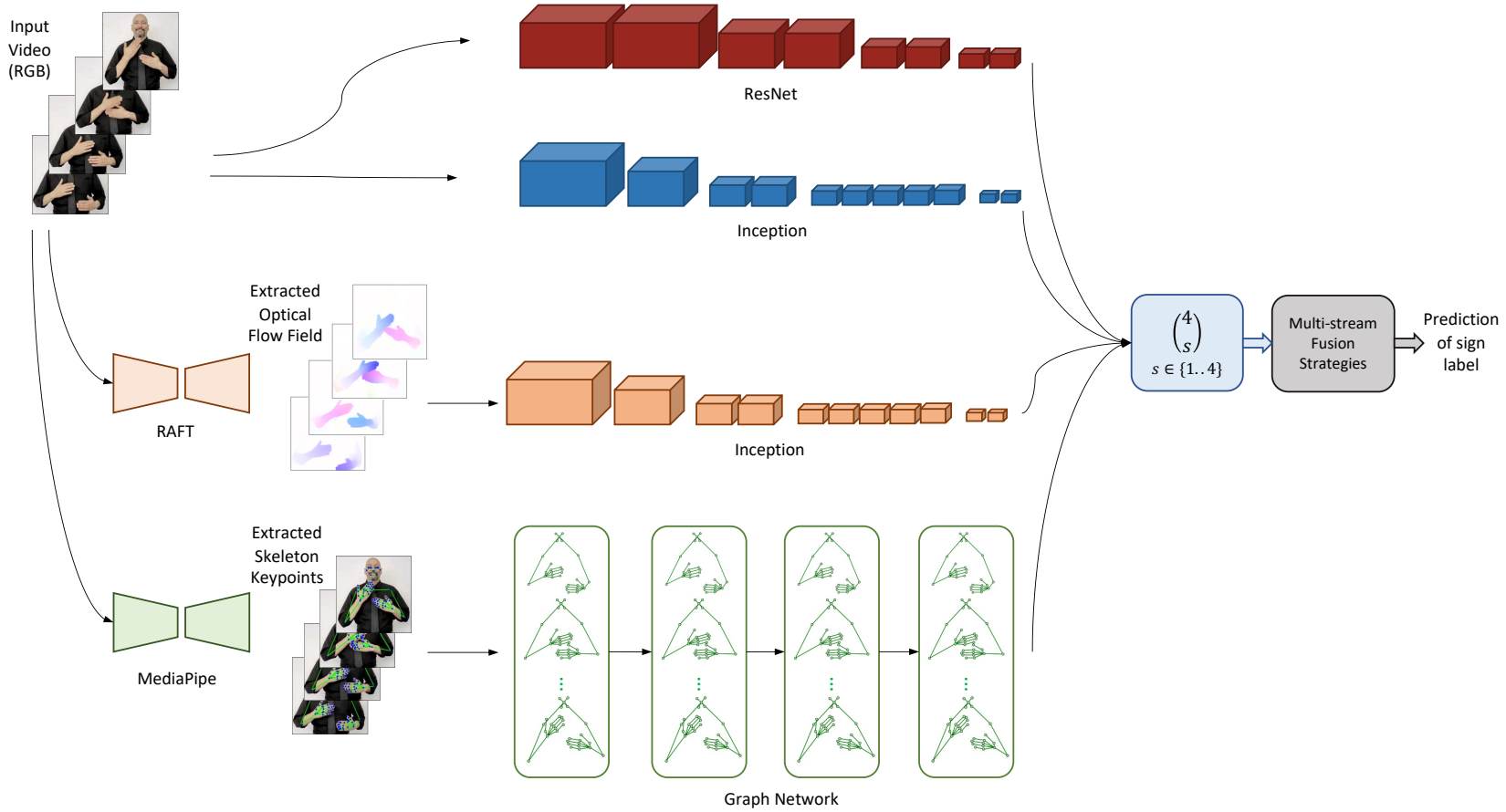


Figure 3.1: The Multi-Stream Framework. Two convolutional models (ResNet and Inception) process the raw RGB input clip, and a similar Inception model operates on the optical flow extracted using RAFT. A skeleton-based model (e.g., RNN, Transformer Encoder, or ST-GCN) operates on the extracted skeleton data of the body and hands. Various combinations of streams and fusion strategies are evaluated.

- **Single output prediction**. The output features or predictions of different streams in the framework are fused to produce a single prediction for the label (class) of the sign performed in the input video. Stream fusion methods should not be confused with temporal fusion methods within a single model, such as late or slow fusion.

Our plan to address the research questions presented in Chapter 1 is as follows.

First Phase

We construct, train, evaluate, analyze, and compare the following individual models.

1. **RGB models**: These includes slow fusion models (ResNet3D and Inception3D) and late fusion models (ResNet2D18, ResNet2D18+GRU, and ResNet2D50) to assess the significance of 3D ConvNet over 2D ConvNet for sign language recognition.

2. **Optical flow model**: An Inception3D model adapted for optical flow input.

3. **Skeleton models**: These include recurrent neural network (GRU), Transformer Encoder, and Graph Convolutional Network (GCN).

This chapter provides the design and training details for the aforementioned models. Their test results and comparative analysis are presented in Chapter 4 (Results and Analysis).

Second Phase

1. **Model Selection**: Based on the validation results in the first phase, we select the best-performing model from each input modality. For RGB, we choose two models so that we can establish a two-stream unimodal baseline. In total, four models will be selected.

2. **Stream Combination Evaluation**: Using the four selected models, we evaluate all possible combinations of: two streams (${}^4C_2 = 6$), three streams (${}^4C_3 = 4$), and all four streams combined.

All aspects of the second phase are covered in Chapter 4, as they focus on results. However, we include our proposed stream fusion methods at the end of this chapter to determine the best approach for combining outputs from multiple streams.

3.2 Raw RGB Models

Deep convolutional neural networks have shown to be very successful for large-scale image recognition [38, 44–46]. In Chapter 2, we reviewed and compared several 3D convolutional models for action recognition in videos, including C3D [1] which is based on VGG architecture [44], and I3D [2], which is an inflated version of Inception2D [47]. For our RGB models, we use convolutional neural networks with ResNet and Inception backbones. The details of these architectures and our modifications are provided in the following sections.

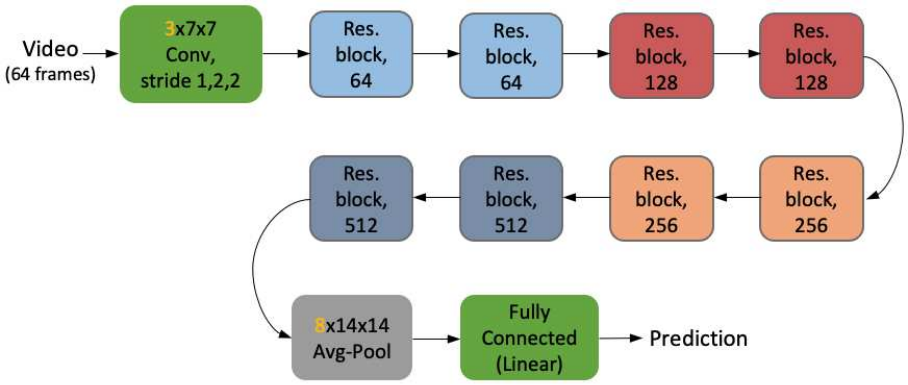
3.2.1 ResNet Backbone

Residual neural networks (ResNet) utilize skip connections to enable the gradient to flow more easily through the network. This design addresses the degradation problem, where the training accuracy degrades as the depth of the neural network increases [45]. Moreover, it creates a smoother loss landscape [74] and helps mitigate the vanishing gradient problem during backpropagation, all of which contribute to better optimization. The simplest version of ResNet (ResNet-18) has 18 layers, which are structured into eight residual blocks. Each residual block consists of two convolutional layers and a skip (shortcut) connection between the input and the output of the residual block (Figure 3.2 b). The residual block is expressed mathematically as follows:

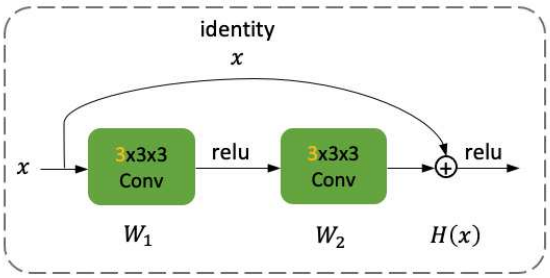
$$H(x) = W_2 * \sigma(W_1 * x) + x \tag{3.1}$$

where x is the input to the block, σ is the activation function (ReLU), W_1 and W_2 are the weights of the two convolutional layers, and $*$ denotes the convolution operation. If the weights W_1 and W_2 are zero, then $H(x) = x$ represents the identity mapping where the block is effectively skipped. In other words, the residual block learns the difference (or residual)

between the input and the output, which is often simpler to learn than mapping the input directly to the output. The number of feature maps (channels) in ResNet-18 is doubled every two residual blocks while their spatial dimensions are halved. This design enables the network to learn more complex features at deeper layers and reduce computational bottlenecks.



(a) Inflated ResNet-18 (R3D)



(b) Inside R3D Residual Block

Figure 3.2: Inflated ResNet-18 (R3D) architecture. The number of filters in each residual block doubles every two blocks. All kernels in the original ResNet-18 2D model [45] are inflated into 3D by adding a temporal dimension (denoted in yellow).

To establish a 2D convolutional network baseline, we modify the ResNet-18 architecture for video input by adding a third dimension of size 1 to all 2D convolutional kernels. This way, 2D convolutions are performed on each frame individually, but not across time. Global average pooling—across both space and time—is applied at the end before the classification layer. We refer to this variant as R2D18 and pretrain it on the Kinetics dataset (Figure 3.3a). Additionally, we develop another variant, referred to as R2D18+GRU, where instead of global average pooling, the outputs from ResNet are fed directly into a GRU layer of 512

hidden units to match the channel dimension of the penultimate layer in ResNet. The final outputs from the GRU are averaged across time before being fed into the classification layer (Figure 3.3b).

A deeper version of ResNet (ResNet-50) has 50 layers structured into 16 residual blocks. Each residual block in ResNet-50 consists of a 1×1 convolution to reduce the number of channels, followed by a 3×3 convolution to process the spatial features, and then a 1×1 convolution to restore the number of channels. This design reduces the computational cost of the 3×3 convolution operations across the 50 layers. We use a 2D version adapted for video by Wang et al. [9], as it is pretrained on the Kinetics dataset. We refer to this 2D variant as R2D50. Similar to R2D18, all kernels in R2D50 have a temporal dimension of size 1 (i.e. no convolution across time) and a global average pooling is applied at the end (Figure 3.3c). However, max pooling across time is applied after the fourth layer in R2D50 to reduce computation and memory.

A 3D adaptation of ResNet18 (referred to as R3D18) inflates the 3×3 convolutional kernels into $3 \times 3 \times 3$ across all layers (Figures 3.2 and 3.3d). R3D18 represents a slow fusion model where temporal features are slowly integrated through its convolutional layers. In contrast, R2D18, R2D18+GRU, and R2D50 represent late fusion models where temporal features are integrated after the last layer, either by averaging (e.g., R2D18, R2D50) or a recurrent layer (R2D18+GRU). We modify the classification layer for all four models in the following three ways.

1. The number of outputs is changed to match the number of classes in the sign dataset.

2. The weights of the classification layer are initialized using Xavier normal distribution [75], and the biases are initialized to zero.

3. We apply Dropout [76] just before the classification layer to randomly drop some output features with a probability of p. This serves as a regularization technique during training to mitigate overfitting. We set the dropout probability to $p = 0.5$ for

R2D50 and $p = 0.2$ for R2D18, R2D18+GRU, and R3D18, because the 50-layer model outputs 2048 features, whereas the others output 512 features.

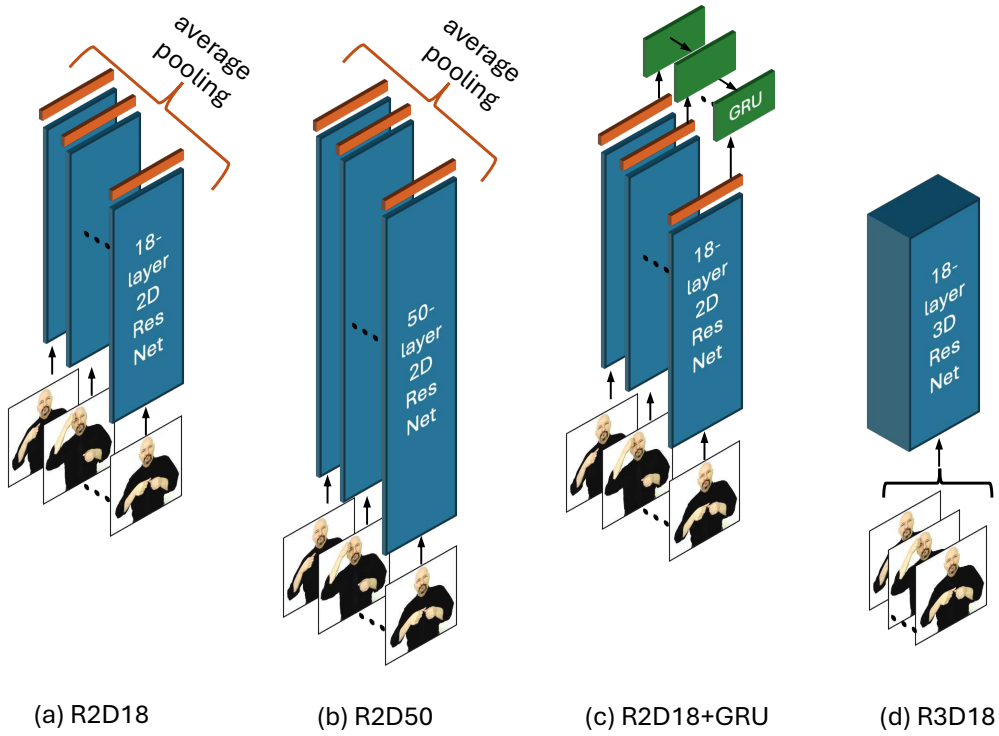


Figure 3.3: The four constructed ResNet variants. (a), (b), and (c) are 2D ConvNets with late fusion of features across time. (d) is a 3D ConvNet (slow fusion).

3.2.2 Inception Backbone

In addition to ResNet-based models, we use the Inception-based 3D convolutional neural network (referred to as I3D) for two reasons. First, I3D is significantly less computationally intensive than R3D18 because it performs more aggressive downsampling of spatiotemporal features in its stem and early layers compared to R3D18. Second, I3D [2] offers pretrained weights for both RGB and optical flow inputs from the Kinetics dataset, whereas R3D18 provides pretrained weights for RGB inputs only. For a fair comparison between optical flow and RGB models, we use the same architecture (inception) and corresponding pretraining.

The original Inception network [46] is a 2D convolutional neural network composed of nine inception blocks. Each block processes the input through parallel paths that apply convolutional filters of different sizes and max pooling operation, then concatenates their outputs across the channel dimension (Figure 3.4 b). Unlike architectures such as VGG and ResNet, which utilize uniform 3×3 convolutions across all layers, the inception architecture is designed to learn features at multiple scales using 1, 3×3, and 5×5 convolutions. This eliminates the need to manually decide the optimal scale. To lower the computational cost of the inception block, a 1×1 convolution is applied before the 3×3 and 5×5 convolutions to reduce the number of channels.

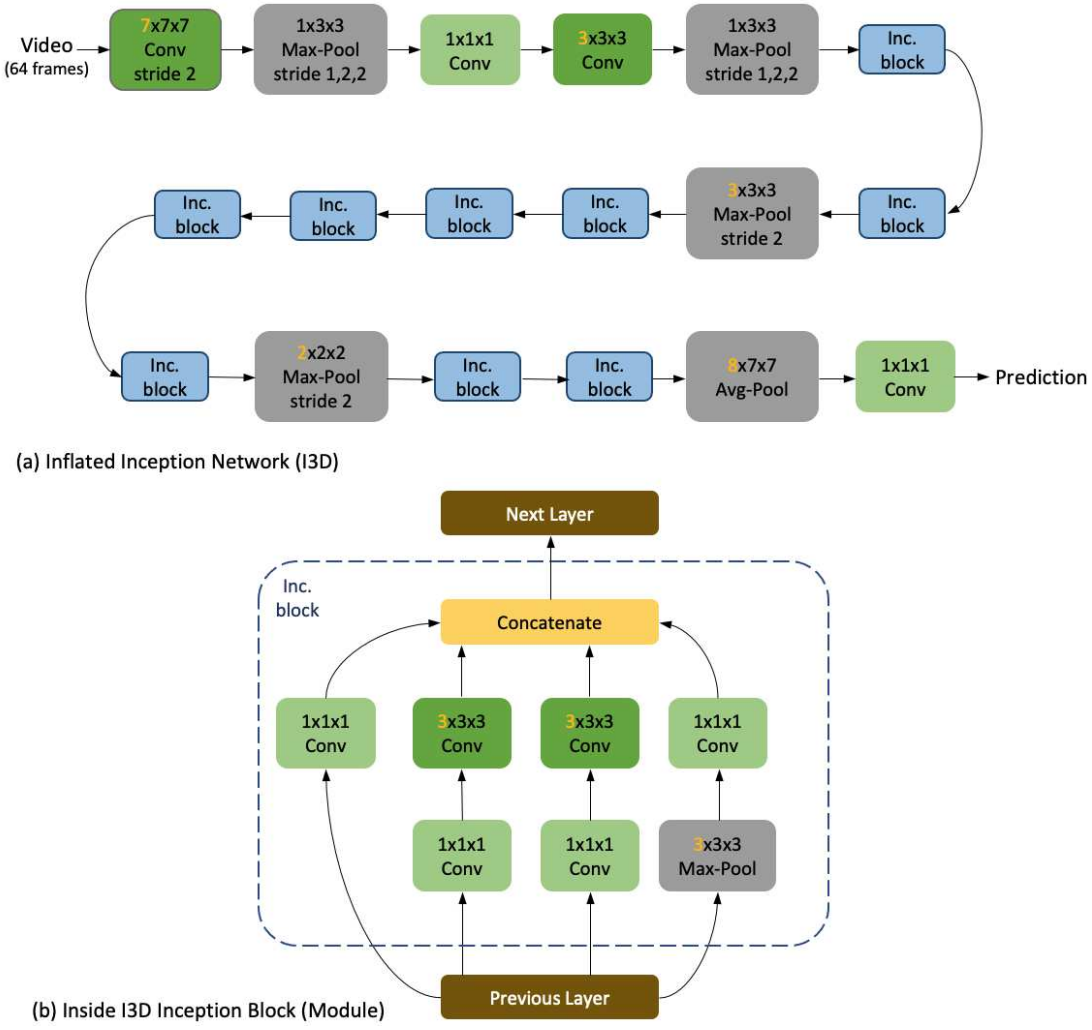


Figure 3.4: Inflated Inception 3D (I3D) architecture [2]. The numbers denoted in yellow are inflated to perform convolution or pooling across time.

Inception 3D (I3D) is the 3D adaptation of Inception 2D architecture. All 2D convolutional kernels and max-pooling are 'inflated' into 3D kernels (Figure 3.4). This is achieved by adding a third dimension of the same size as the first two. For instance, a 3×3 kernel becomes $3 \times 3 \times 3$ and a 1×1 kernel becomes $1 \times 1 \times 1$. While I3D was originally developed in TensorFlow, we use the PyTorch implementation and the pretrained weight imported from TensorFow [77]. We apply the three modifications as described in Section 3.2.1 to adapt the I3D architecture for sign datasets. Since the I3D model outputs 1024 features, we set a dropout probability of $p = 0.5$ before the classification layer as we find this value to work well on the validation set.

Besides ResNet and Inception convolutional architectures, we also explored architectures that employ spatiotemporal separable convolutions such as S3D [3], or depthwise separable convolutions such as ConvNeXt [78]. However, separable convolutions require more memory and did not improve the accuracy relative to ResNet and Inception in our experiments.

3.3 Optical Flow Model

There are two stages within the optical flow model: (i) extracting the optical flow from the original RGB input frames, and (ii) training a convolutional neural network (I3D) to classify the sign based on the extracted optical flow (Figure 3.1). The description of each stage is as follows.

First Stage

An optical flow between two image frames represents the apparent motion of individual pixels from one frame to the next. More details about the optical flow field and the stacking methods were provided in two-stream networks in Chapter 2 (Section 2.4.1). The methods for extracting optical flow can be broadly classified into two categories: analytical (classical) algorithms and machine learning-based methods. Among classical mehods, TV-L1 algorithm has been widely used in state-of-the-art approaches for action and sign recognition [2, 14, 79].

It is an optimization algorithm based on variational methods that solves a mathematical energy minimization problem to estimate motion between frames [53].

We adopt a neural network-based approach for extracting optical flow using Recurrent All-Pairs Field Transforms (RAFT) [56]. RAFT uses a ConvGRU unit (discussed in Chapter 2, Section 2.5.2) to iteratively refine the optical flow estimation. At each iteration, the ConvGRU unit receives the concatenation of three inputs: (i) the correlation features between two images, (ii) the context features from the first image, and (iii) the current optical flow estimate. The correlation volume and context features are computed once at the beginning, while the current optical flow estimate is iteratively updated and used to lookup and sample local features from the correlation volume for each point. At each iteration, the ConvGRU outputs an updated hidden state, which is then used to estimate the adjustment to the current optical flow estimate. To reduce computational cost during the iterations, the features and optical flow are computed at a downsampled resolution, and the final optical flow is upsampled to match the original input resolution. To extract optical flow from a video, we apply this process between every two consecutive frames.

Figure 3.5 shows the optical flow extracted by large RAFT, small RAFT, and TV-L1 for a sample frame. The large RAFT model produces significantly more accurate optical flow compared to the small RAFT and TV-L1. However, this improved accuracy comes at a higher computational cost.

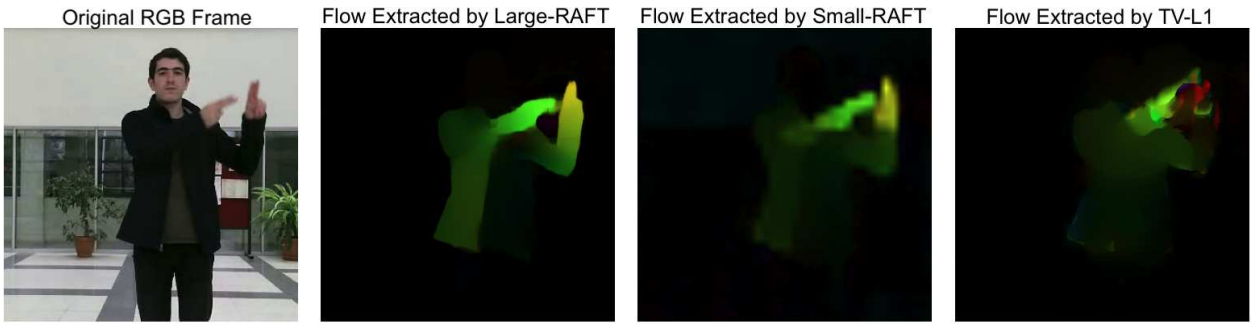


Figure 3.5: Comparison of the optical flow quality extracted by the large RAFT, small RAFT, and TV-L1 from a sample RGB input.

To estimate the computational cost of RAFT, we used FlopCountAnalysis function provided by the fvcore library [80]. For an input video with 64 frames and 256×256 spatial size, the large RAFT uses 3.3 TFLOPs while the small RAFT uses 740 GFLOPs. To quantify the inference speed of these methods, we benchmarked their performance on an NVIDIA RTX 3090 GPU. The large RAFT achieved 36 FPS, while the small RAFT achieved 45 FPS. On a CPU, the frame rates were significantly lower: the large and small RAFTs achieved 3 FPS and 8 FPS, respectively, and TV-L1 achieved about 3 FPS. Despite the higher computational cost of the large RAFT, we adopt it in the first stage of our optical flow pipeline so that our conclusion about using optical flow in sign language recognition is based on a best-case scenario of high-quality optical flow.

The large RAFT model we use in the first stage was pretrained by the authors on the FlyingChairs and FlyingThings datasets and finetuned on Sintel, KittiFlow, and HD1K datasets [56]. During training on sign recognition datasets, we keep the RAFT model's weights frozen and update only the weights of the model in the second stage.

Second Stage

In the second stage, the optical flow extracted from every two consecutive frames (by large RAFT) is fed into an inception-based 3D convolutional neural network (I3D) for sign prediction. The I3D model we use for the optical flow input is identical to the one used for RGB input, as described in Section 3.2.2, with two key distinctions: (i) the first layer of this model expects an input with two channels, representing the horizontal and vertical components of the optical flow field, instead of three channels for RGB colors, and (ii) this model is initialized with weights pretrained on the optical flow of the Kinetics dataset. All other aspects and modifications mentioned in Section 3.2.2 such as dropout remain the same. To differentiate between the I3D model with RGB input and the I3D model with optical flow input, we refer to the latter as F3D throughout this document.

3.4 Skeleton Models

Skeleton data represents the set of keypoints (joints) and connections (bones) in the human body. There are two stages within a skeleton model: (i) extracting the skeleton from the RGB input frames, and (ii) training a model to classify the sign based on the extracted skeleton data (Figure 3.1). The description of each stage is as follows.

First Stage

Pose estimation refers to the process of extracting the coordinates of body keypoints from visual input, which can serve as the first stage in an action recognition pipeline. Pose estimation typically detects key landmarks on the human body, such as shoulders, elbows, wrists, lips, and eyes, which are critical for sign recognition. However, sign recognition is a fine-grained classification problem that depends heavily on hand configuration, orientation, and location. Therefore, in addition to body landmarks, the keypoints of both hands must be included in the skeleton data used for sign recognition. Figure 3.6 illustrates the body and hand landmarks extracted by MediaPipe's pose and hand modules, respectively.

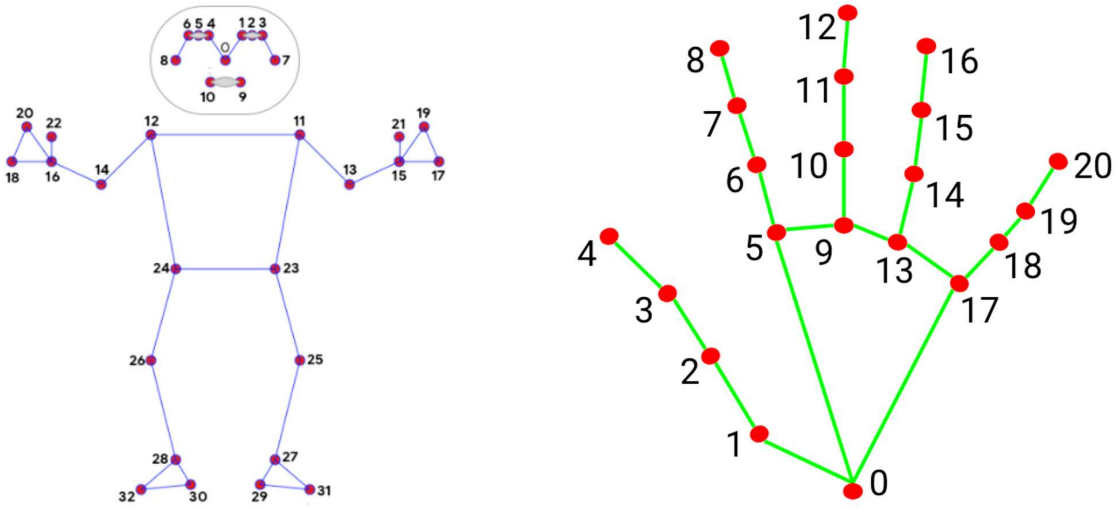


Figure 3.6: Body and hand landmarks as defined by MediaPipe's pose and hands modules, respectively. Source: [81, 82].

We use MediaPipe to extract the skeleton data because it is designed for real-time use on mobile devices. While more computationally expensive alternatives like HRNet [33] provide more accurate skeleton estimation, we chose MediaPipe for its real time performance. In contrast to the optical flow model, where we opted for a more accurate but resource-intensive method (large RAFT) in the first stage, we prioritized a lightweight approach for skeleton extraction to demonstrate that, despite this trade-off, skeleton-based sign recognition still outperforms the optical flow model in both single- and multi-stream frameworks.

We use two modules in MediaPipe: the pose module which extracts 33 body keypoints (Figure 3.6 a), and the hands module which extracts 21 keypoints from each hand (Figure 3.6 b). MediaPipe's pose module is based on BlazePose [81] which utilizes an encoder-decoder network to predict pose heatmaps, followed by another regression encoder network to estimate the coordinates of body keypoints. MediaPipe's hands module [35] is based on convolutional pose machines [20, 83], which use a multi-stage refinement process. In its first stage, it predicts initial heatmaps for each hand keypoint. In subsequent stages, these predictions are refined using the heatmaps from the previous stage and extracted image features.

Out of the 33 body keypoints shown in Figure 3.6 a, we select the points corresponding to the nose (0), eyes (2 and 5), mouth (9 and 10), shoulders (11 and 12), and elbows (13 and 14). We omit the keypoints from the lower body (e.g. knees and feet) as they are not relevant for sign recognition. From the 21 hand keypoints in Figure 3.6 b, we select the wrist (0), the bases of the five fingers (1, 5, 9, 13, and 17) and the tips of the five fingers (4, 8, 12, 16, 20) on both hands. In total, we utilize 31 keypoints for skeleton-based sign recognition.

In addition to the selected keypoints, we also define connections between them to follow a tree structure resembling the connectivity of the human skeleton, with few exceptions. We consider the nose to be the root node, which is connected to six nodes: two for the eyes, two for the mouth, and two for the shoulders. Each shoulder joint is connected to the

corresponding elbow, which is then connected to the wrist. Each wrist is connected to the five finger bases of the same hand, which in turn are connected to the respective finger tips.

We represent the skeleton input to the second stage using five channels. The first and second channels correspond to the x and y coordinates of the 31 keypoints, respectively. The third and fourth channels correspond to the horizontal and vertical distances, respectively, of each connection between connected keypoints as described above. Specifically, for a connection between keypoints i and $i+1$, the horizontal distance is given by $(x_{i+1}–x_i)$, and the vertical distance is given by $(y_{i+1} - y_i)$. If a keypoint is not detected—either due to a false negative by MediaPipe or because the keypoint is out of the frame—we set its coordinates (x, y) to $(0, 0)$. To differentiate between $(0, 0)$ as a valid coordinate and as a placeholder for failed detection, we introduce a fifth channel to encode detection reliability using a binary flag. A value of 1 in the fifth channel indicates to the model in the second stage that the keypoint was successfully detected, while a value of 0 indicates a failed detection.

In summary, the skeleton input fed into the second stage consists of 31 keypoints for each frame in the video, with each keypoint represented by five values: the x-coordinate, the y-coordinate, the horizontal distance to the next keypoint, the vertical distance to the next keypoint, and the binary detection flag.

For the second stage of the skeleton model, we construct and evaluate three different architectures: recurrent neural network, transformer encoder, and graph convolutional network. The details of these models are provided in Sections 3.4.1, 3.4.2, and 3.4.3, respectively.

3.4.1 Recurrent Neural Network

Recurrent neural networks (RNNs) are a natural choice for skeleton-based sign language recognition due to the sequential nature of video frames. Gated Recurrent Unit (GRU) [63] and Long Short-Term Memory LSTM [60] are the most commonly used RNN variants that address the limitations of vanilla RNNs, such as vanishing or exploding gradients. We briefly

experimented with both GRU and LSTM and found that GRU achieves comparable accuracy to LSTM while having a simpler architecture and better computational efficiency.

Let the input x_t be a vector formed by concatenating the coordinates of all 31 keypoints extracted from frame t. We apply layer normalization on x_t to normalize across its features (channels) before passing it into the GRU unit. At each time step, the GRU unit computes the hidden state h_t as a function of the current input x_t and the previous hidden state h_{t-1}. The final output is obtained by averaging all hidden states across time and feeding the result into a fully connected layer for classification.

We experimented with several hyperparameters of the GRU model, including the size of the hidden state, the number of recurrent layers, and the dropout probability. For the hidden state, we tried sizes of 64, 128, 256, and 512 units. We selected 512 units since no noticeable improvement in validation accuracy was observed beyond this size. We followed a similar process to set the GRU with three recurrent layers and a dropout probability of 0.5 before the classification layer. Additionally, we experimented with a bidirectional GRU variant but it did not improve the accuracy.

3.4.2 Transformer Encoder

The transformer architecture, proposed by Vaswani et al. [65], addresses several limitations in RNNs. Unlike RNNs which are inherently sequential, transformers process all tokens in a sequence simultaneously, making them highly parallelizable. Moreover, transformers capture long-range dependencies more effectively due to the self-attention mechanism which enables the model to directly attend to any part of the input sequence, regardless of the distance. Transformers are more scalable making them better suited for constructing large language models (LLMs) [66, 67].

The transformer encoder model we use consists of several blocks, each containing a multi-head self-attention mechanism followed by a fully-connected layer (FC), along with residual connections and layer normalization (Figure 3.7). In contrast to word embeddings, where

the representation of a token is fixed, the self-attention mechanism introduces contextual embeddings where the representation of a token changes dynamically depending on its surrounding context. During training, the model learns the query, key, and value functions for the input sequence, such that the dot product between a query and a key determines the amount of attention assigned to the value associated with that key, as follows:

$$Attention(Q, K, V) = Softmax(QK^T)V \tag{3.2}$$

where the queries $Q = XW_q$, keys $K = XW_k$, and values $V = XW_v$ are computed from the input sequence X, and W_q, W_k, W_v are learnable weight matrices. The Softmax function is used to normalize the computed attention scores QK^T. The output of the self-attention layer at a time step t is a weighted sum of the values across all time steps, where the weights are determined by the similarity between the query at time t and all keys in the sequence.

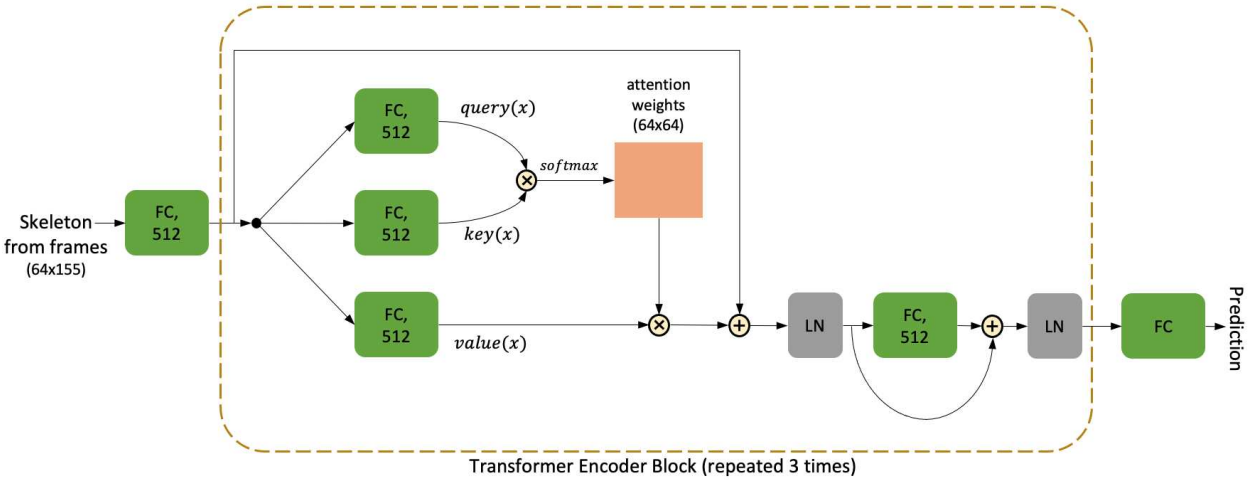


Figure 3.7: Transformer encoder architecture for skeleton data, based on [65]. FC stands for fully-connected (linear) layer and LN for layer normalization.

Each token in the input represents the skeleton data for a single frame, which consists of 31 keypoints each with 5 channels resulting in 155 values. We first project the input dimension from 155 to 512 using a fully connected layer, which is then passed through the transformer

encoder. The final output is obtained by averaging the token representations from the last layer across time and feeding the result into a fully connected layer for classification.

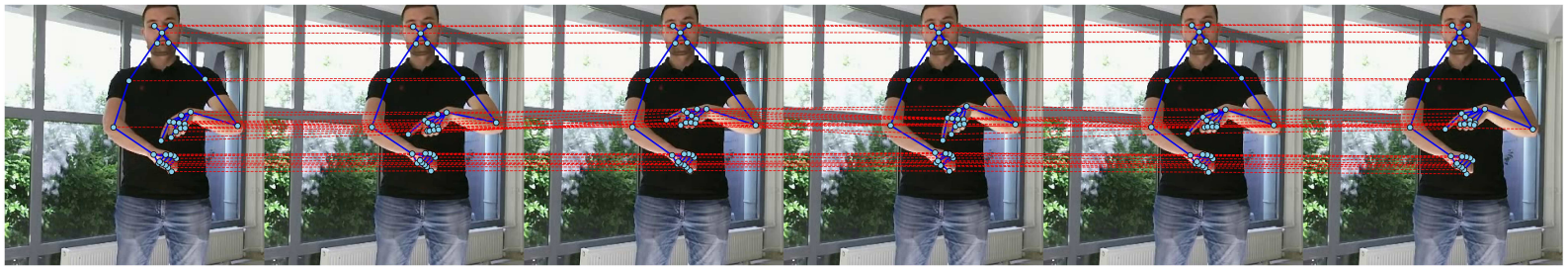
Based on the model's performance on the validation set, we selected the following hyper-parameters: a feature dimension of $d = 512$, a dropout probability of 0.1, four self-attention heads, and three encoder blocks. We briefly experimented with positional encoding, as described in [65], to encode the ordering of the frames in the skeleton input but it did not improve the model's accuracy for sign recognition.

3.4.3 Graph Convolutional Network

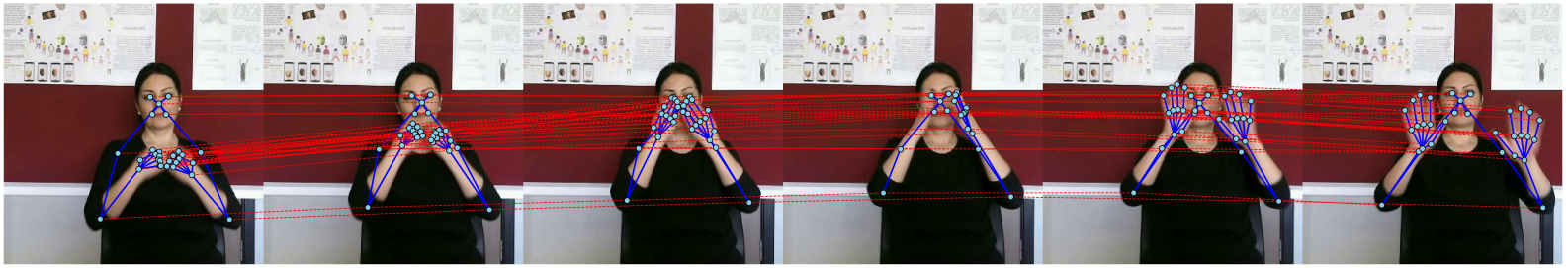
Graph convolutional networks (GCNs) are a well-suited architecture for skeleton-based sign language recognition because they enable the use of prior knowledge about the skeleton structure by representing it as a graph, where joints are represented as nodes and bones as edges connecting these nodes. In contrast, RNNs and the transformer encoder process the skeleton data as a flattened input, where each frame is represented as a bag of features consisting of 155 values (31 keypoints $\times$ 5 channels).

Graph convolutional network (GCNs) [84] extend the concept of convolution, which is highly effective for grid-structured data such as images, to graph-structured data. However, a standard GCN operates on a static graph, where the relationships between nodes (e.g. distance) do not change over time. The Spatio-Temporal GCN (ST-GCN) [85] addresses this limitation by incorporating both spatial edges between nodes within a frame and temporal edges connecting the same node across frames (Figure 3.8). We chose the ST-GCN architecture as it is a better fit for the sequential and graph-like nature of the skeleton data. In this document, we use GCN to refer to ST-GCN for brevity, unless stated otherwise.

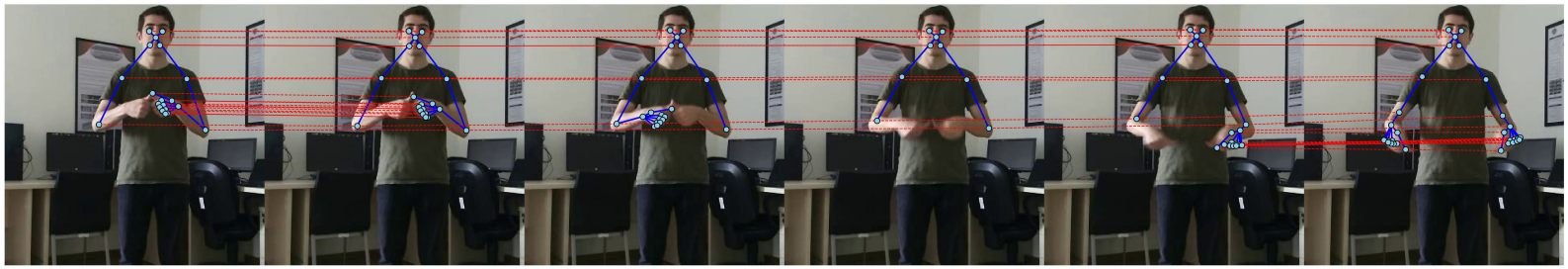
ST-GCN consists of 10 layers, each performing spatial and temporal convolutions on the graph. In spatial convolution, the feature representation of each node is updated by aggregating information from its neighbors in the same frame.



Graph representation for a "time" sign (class #224)



Graph representation for a "garden" sign (class #21)



Graph representation for a "band-aid" sign (class #210) - hand joints are somethimes not detected (false negative)

Figure 3.8: Visualizations of the spatiotemporal graph representation of skeleton data. The spatial edges (blue) connect different joints within each frame, while the temporal edges (red) connect each joint to itself across frames.

Following the implementation in [85, 86], this update operation can be reformulated as follows:

$$H^{(l+1)} = \sigma\left(\sum_{k=1}^{K}\left(A_k D_k^{-1}\right) \odot E_k^{(l)} H^{(l)} W_k^{(l)}\right) \tag{3.3}$$

In this equation:

- $H^{(l)} \in \mathbb{R}^{N \times d_l}$ is the input node feature matrix at layer l, where N is the number of nodes, and d_l is the input feature dimension.
- $H^{(l+1)} \in \mathbb{R}^{N \times d_{l+1}}$ is the updated node feature matrix at layer $l+1$, where d_{l+1} is the output feature dimension.
- $A_k \in \mathbb{R}^{N \times N}$ is the adjacency matrix for partition k, normalized using the degree matrix $D_k \in \mathbb{R}^{N \times N}$ to ensure that the sum of all incoming edges for each node equals 1.
- $E_k^{(l)} \in \mathbb{R}^{N \times N}$ is the learnable edge importance matrix used to weight the significance of each edge in the graph since not all edges are equally important.
- $W_k^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ is the learnable weight matrix for partition k at layer l.
- σ is the activation function (ReLU).
- $\odot$ denotes element-wise product.

Partitioning a graph for spatial convolution in Eq 3.3 involves dividing its nodes into K subsets (partitions), where each partition is associated with its own weight W_k. This approach addresses the irregular structure of graphs, where nodes can vary in both the number and arrangement of their neighbors, unlike the uniform and regular grid structure of images. We adopt the spatial partitioning strategy as proposed by [85] to divide the graph into three partitions ($K = 3$): i) the node itself, ii) neighboring nodes closer to the skeleton's gravity center than the node, and iii) neighboring nodes further from the gravity center than the node. We define the skeleton's gravity center as the nose's coordinate. We also experimented with alternative partitioning strategies but observed suboptimal results.

While spatial convolution is applied independently to each frame's skeleton to capture spatial relationships, temporal convolution is performed subsequently to integrate information across frames and capture motion. In temporal convolution, the feature representation of each node is updated by aggregating information from the same node in neighboring frames using a 1D convolution as follows:

$$Z^{(l+1)} = \text{Conv1D}\left(Z^{(l)}, W_{\text{temporal}}\right) \tag{3.4}$$

In this equation:

- $Z^{(l)} \in \mathbb{R}^{T \times N \times d_l}$ is the input node feature matrix at layer l, representing the output of the spatial convolution $H^{(l+1)}$ across all frames. Here, T is the number of time steps (i.e. frames), N is the number of nodes, and d_l is the input feature dimension.
- $Z^{(l+1)} \in \mathbb{R}^{T' \times N \times d_l}$ is the updated node feature matrix after convolution, where T' is the number of output time steps.
- $W_{\text{temporal}} \in \mathbb{R}^{s_t \times d_l \times d_l}$ is the learnable weight matrix for the temporal convolution, where s_t is the temporal kernel size.

Setting $s_t = 1$ limits the temporal receptive field to a single frame, effectively eliminating convolution across time — equivalent to removing the red links in Figure 3.8 between each node and itself in the next frame. We set the temporal kernel size $s_t = 9$ to provide a large enough window for aggregating information from neighboring frames. In each spatiotemporal layer of the ST-GCN, a dropout probability of 0.5 is applied to the output of the temporal convolution, and a residual connection is added by summing the input with the output, followed by a ReLU activation. The final output of the ST-GCN model is obtained by averaging the 256 features from the last layer across all nodes and time steps and then feeding the result into a fully connected layer for sign classification.

3.5 Training Pipeline for RGB and Flow Models

The details of the training pipeline described in this section apply to the RGB models, including I3D and the four variants of Resnet: R2D18, R2D18+GRU, R2D50, R3D18, and to the second stage of the optical flow model (F3D). While the training process generally follows similar steps, specific differences for datasets will be highlighted in context.

3.5.1 Loading and Resizing Frames

Video files selected in the mini-batch are first loaded from the disk using the Decord video reader [87], as it is faster than OpenCV. All frames in the video are resized to a fixed size of 256×256 pixels for WL-ASL and MS-ASL datasets. Since all videos in AUTSL are of the same size, 512×512 pixels, they are loaded with the original resolution. We experimented with lowering this loading resolution, but that negatively affected the classification accuracy.

3.5.2 Data Augmentation

All loaded video files are randomly cropped both temporally and spatially. For temporal cropping, only 64 consecutive frames are randomly sampled from each video, which is close to the average number of frames per video in the datasets. There are two cases: if the number of frames in the video is 64 or more, the starting frame t_1 is uniformly sampled from the range [0, nframes - 64], where nframes is the total number of frames in the video. If the video has fewer than 64 frames, we experimented with three strategies: filling the difference with frames of zeros, filling the difference by repeating the last frame of the video, or looping the video from the start until there are 64 frames. We found the third strategy to work slightly better in most cases. This choice also aligns with the observation that many signers in longer videos tend to repeat their signs more than once.

After the frames in the video have been selected and resized, we apply a random spatial cropping. For MS-ASL and WL-ASL datasets, a random crop of size 224×224 pixels is uniformly sampled from the loaded 256×256 video. This covers about 76% of the resized

frame which is sufficient to capture at least part of the hands and face (Figure 3.9a). It also matches the expected input size for the original 2D ResNet and Inception models developed for the ImageNet dataset, from which the inflated 3D versions were developed.

For videos in the AUTSL dataset, a spatial crop of size 256×256 pixels is sampled from the original 512×512 video. Since this crop size covers only 25% of the frame, there is a risk the model might learn features unrelated to the sign in the video, such as the background elements, especially if the random crop occurred near the edges. To address this, other approaches use a bounding box extracted by a pose detector to crop a 256×256 region that covers the signer's face and hands [88]. However, for the purpose of this dataset, we believe this adds unnecessary computational overhead to the RGB pipeline, as an additional neural network is required to process the raw RGB input.

(a) A 224×224 crop sampled uniformly from 256×256 resized frame

(b) A 256×256 crop sampled with a normal distribution from 512×512 frame

Figure 3.9: Random cropping strategies used for data augmentation in training. (a) Applied to MSASL and WLASL datasets. (b) Applied to the AUTSL dataset.

We propose a solution that avoids computing bounding boxes based on the observation that recorded signers typically perform their signs in the center of the frame. Therefore, instead of sampling the 256×256 crop uniformly from the 512×512 region, our solution

samples X and Y coordinates of the upper left corner of the crop from a truncated normal distribution with a narrow standard deviation to ensure most of the sampled crops are closer to the center than the edges (Figure 3.9b). This is expressed mathematically as follows:

$$X = 128 + 64 \cdot \text{clip}(\mathcal{N}(0, 0.5), -2, 2) \tag{3.5}$$

$$Y = 128 + 64 \cdot \text{clip}(\mathcal{N}(0, 0.5), -2, 2) \tag{3.6}$$

where $\mathcal{N}(0, 0.5)$ represents a normal distribution with a mean of 0 and a standard deviation of 0.5. The clip function truncates random values beyond -2 and 2 to ensure the frame edges are not crossed. The result is scaled by 64 and shifted by 128 to obtain pixel coordinates within the $[0, 256]$ range.

As a final step in the data augmentation, frames in the crop are flipped horizontally with a 0.5 probability for MS-ASL and WL-ASL datasets, as signs can be performed with either hand. We reduce this probability to 0.25 for AUTSL, since only 2 out of 43 signers are left-handed [28].

3.5.3 Batch Size and Normalization

After the crops are sampled from files selected for the minibatch, the batch loader groups them together into a single five-dimensional array of size (N, T, H, W, C), where N is the batch size, T=64 is the clip length (i.e., the number of frames), H=256 is the crop height, W=256 is the crop width, and C is the number of channels (3 for RGB or 2 for flow).

In our experimentation, we found that the batch size must be chosen carefully, especially for unbalanced datasets such as WLASL and MSASL, as it can significantly impact the generalization accuracy. We observe that small batch sizes yield much better generalization accuracy with a wide margin. Our hypothesis is that in an unbalanced dataset, minority classes are harder to learn in a large batch size since the batch is more likely to be dominated

by majority classes. On the other hand, using a small batch size for AUTSL was not as helpful since it is a balanced dataset.

Each batch is normalized before being passed into the model with the mean and standard deviation of the dataset used for pretraining the model. For RGB models, we use the following mean μ and standard deviation σ from ImageNet to normalize the three input channels:

$$\mu = [0.485, 0.456, 0.406], \quad \sigma = [0.229, 0.224, 0.225] \tag{3.7}$$

The normalized batch is then transposed to (N, C, T, H, W), the shape of the input array expected by the convolutional model.

3.5.4 The Learning Rate

The learning rate is a critical hyperparameter that requires careful tuning, as it can affect the model's accuracy and convergence speed. In this section, we describe our approach to finding a suitable learning rate before the start of training and scheduling the learning rate once the training begins.

Finding the learning rate

To minimize trial-and-error in the search for a learning rate, we apply the systematic approach described in [89] and [90] to find the highest learning rate to minimize the loss before the loss rapidly spikes upwards. Figure 3.10 shows the loss versus learning rate in finding the learning rate experiment for the I3D model on AUTSL dataset using Adam [73] as the optimizer.

The process of finding such a learning rate starts by training the model on a very small initial learning rate, such as $lr_0 = 10^{-8}$, and exponentially increasing it up to a large value, $lr_{n-1} = 10$. At each time step, the learning rate is updated by multiplying it by a fixed factor of $(lr_{n-1}/lr_0)^{1/n-1}$, where n is the number of time steps in one epoch. The loss usually spikes to a very high value before the epoch ends.

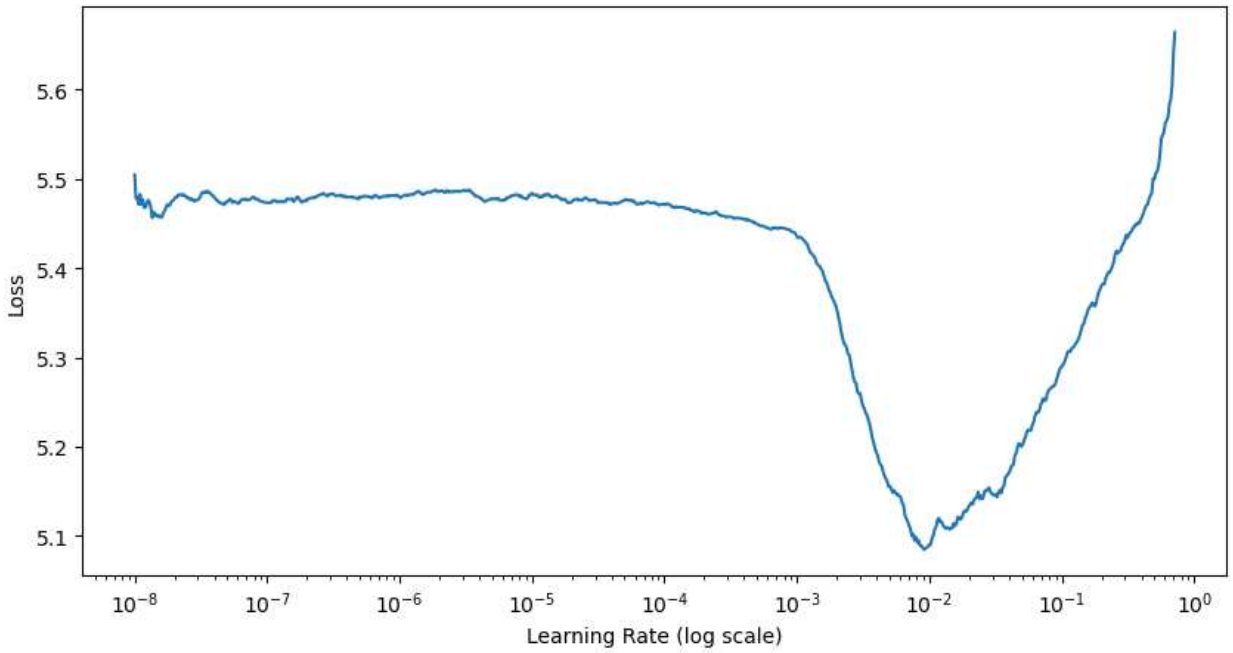


Figure 3.10: Finding the learning rate for training I3D on AUTSL

For better visualization, the loss shown in Figure 3.10 is smoothed by calculating the exponentially weighted averages of all losses in previous time steps. This can be expressed by the recurrence formula [90]:

$$\text{avg loss}_t = \beta \times \text{avg loss}_{t-1} + (1 - \beta) * \text{loss}_t \tag{3.8}$$

$$\text{smoothed loss}_t = \text{avg loss}_t / (1 - \beta^{t+1}) \tag{3.9}$$

where β is the decay factor set to be 0.98 in our case. The average loss is divided by the bias correction factor $1 - \beta^{t+1}$ to prevent the loss from being underestimated in early time steps.

To pick our learning rate from Figure 3.10, note that the smoothed loss for I3D reaches its minimum value at 1×10^{-2} then rapidly spikes upwards. However, since the shown loss is a weighted average, the actual loss at 1×10^{-2} is likely to be too high already for the model. Therefore, we adopt the heuristic proposed in [90] to choose our learning rate for the model as one order of magnitude less than the learning rate of the minimum smoothed loss, which

equals 1×10^{-3} in this case. Following a similar procedure for R3D, we found the optimal learning rate on AUTSL to be around 3×10^{-4}.

Scheduling the learning rate

In our experiments, reducing the learning rate periodically during training has consistently improved the validation accuracy across all sign datasets. Several methods have been proposed for scheduling the decay of the learning rate during training, such as step reduction every fixed number of epochs, step reduction when validation accuracy plateaus, and cosine annealing [91].

Moreover, warm-up scheduling, such as linearly increasing the learning rate at the start of training until it reaches a maximum value, has shown better convergence for adaptive stochastic optimization techniques such as Adam [65] based on the hypothesis that warm-up acts as a variance reduction technique [92].

Our approach for scheduling the learning rate combines both phases of warming up and decaying the learning rate using a one-cycle schedule [93]. In the warm-up phase, which constitutes the first 30% of the training steps, the learning rate starts at a fraction of the maximum learning rate ($lr_{max}/25$) and gradually increases at each training step with a cosine schedule until it reaches lr_{max} (e.g. 1×10^{-3} for I3D). Then, the learning rate gradually decreases for the remainder of the training until it reaches a very small value of $lr_{max}/(25 \times 10^4)$ by the end of training. The top plots of Figures 3.11 and 3.12 illustrate the one-cycle learning rate schedule applied to Adaptive Moment Estimation (Adam) and Stochastic Gradient Descent (SGD) with momentum, respectively, on WL100 dataset.

In both figures, we compare the generalization performance of the one-cycle learning rate schedule against the commonly used reduce-on-plateau schedule. In our training experiment, using Adam and a one-cycle schedule improved the validation accuracy over Adam with a reduce-on-plateau from 75.7% to 79% (Figure 3.11). Using SGD with momentum underperformed Adam in both one-cycle and reduce-on-plateau schedules (Figure 3.12), achieving only about 72% validation accuracy. Our experiment in Figure 3.11 supports the hypoth-

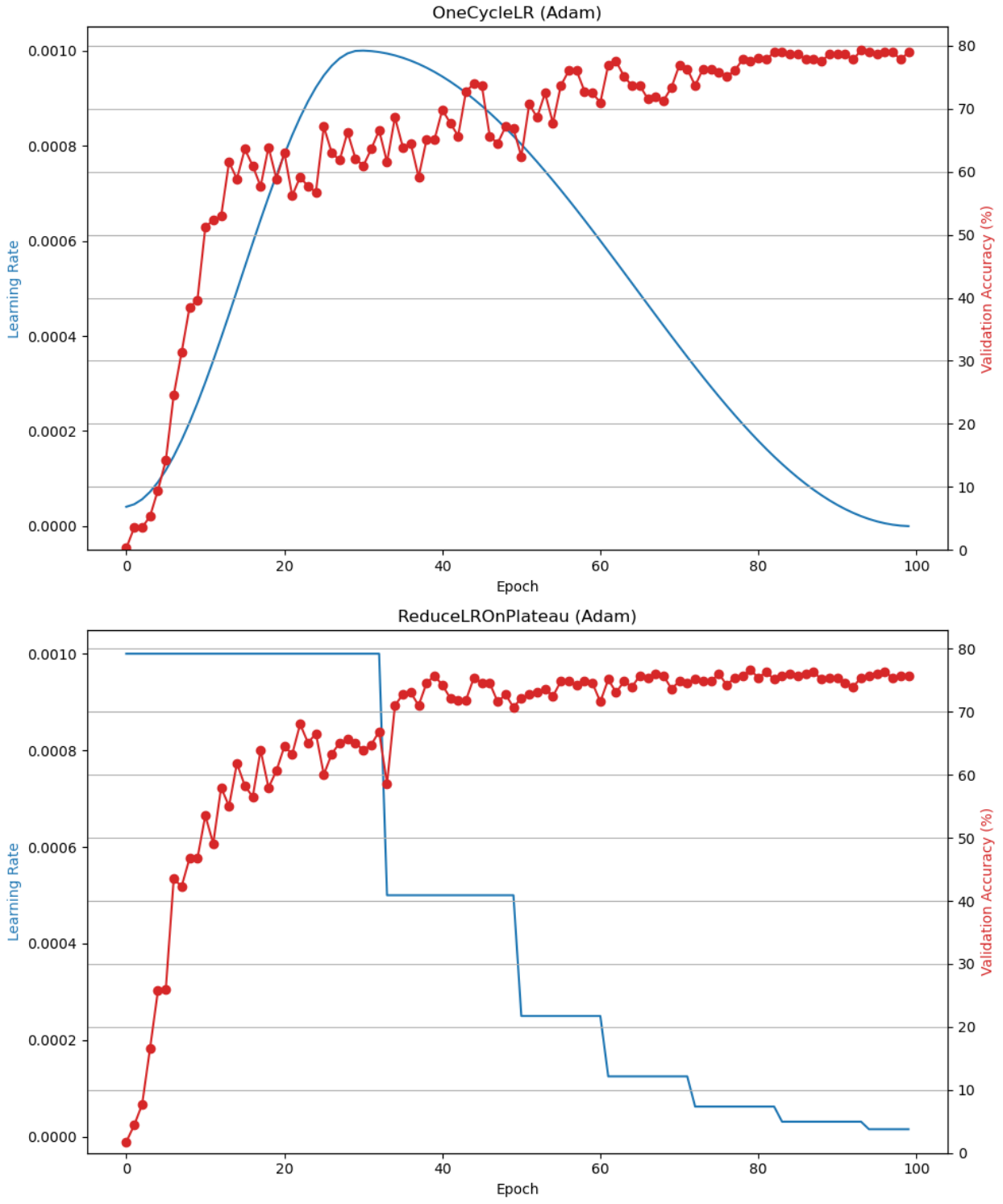


Figure 3.11: Scheduling the Learning Rate: One-Cycle and Reduce on Plateau for Adam optimization on WL100

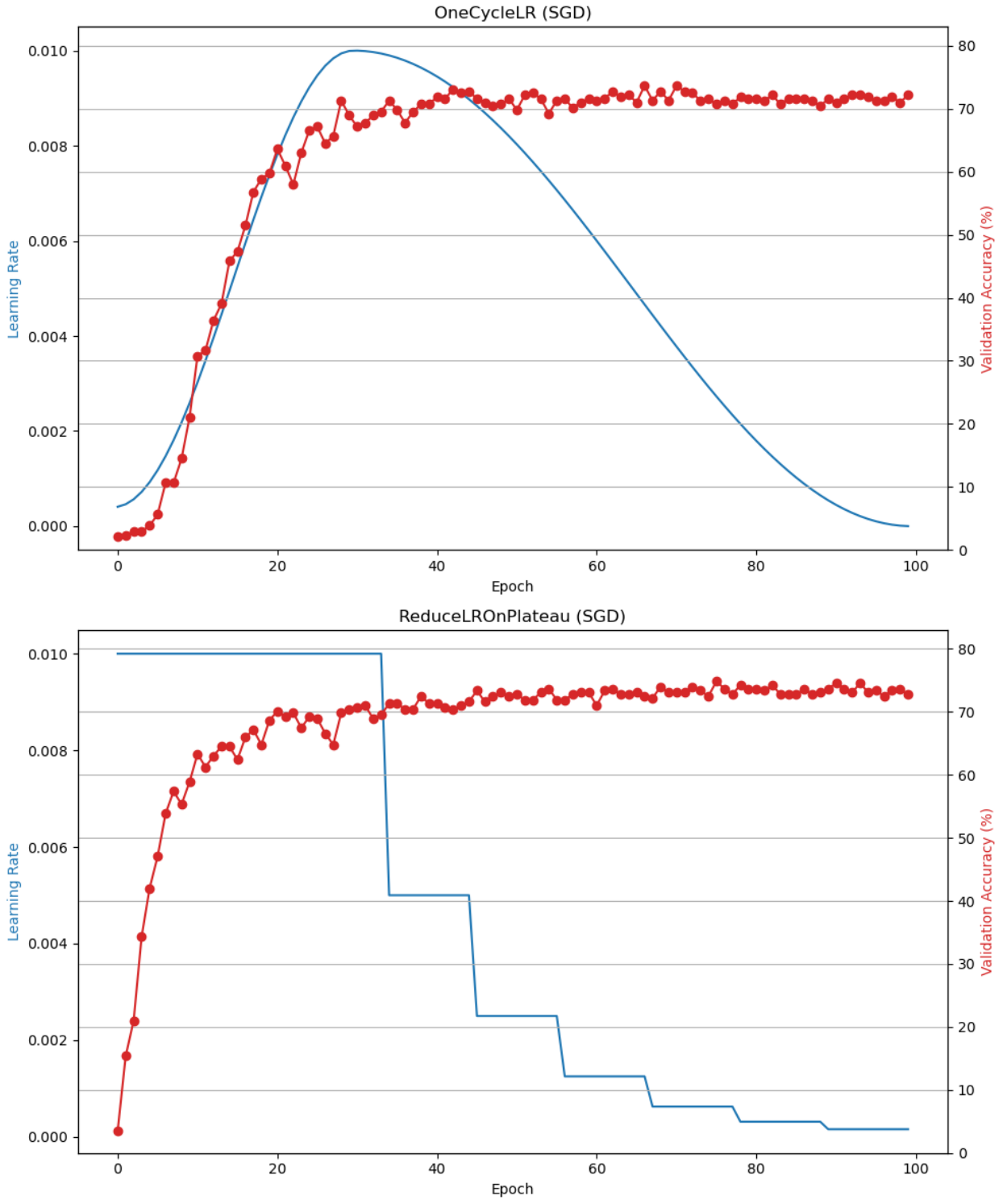


Figure 3.12: Scheduling the Learning Rate: One-Cycle and Reduce on Plateau for SGD optimization on WL100

esis proposed by [93, 94] that approaching the maximum high learning rate, resulting in a more volatile validation accuracy in the middle of a one-cycle policy, acts as a regularization method and thus helps mitigate overfitting and improve generalization accuracy. Using one-cycle policy with Adam optimization and Dropout regularization provided sufficient regularization for training our RGB and flow models. We also experimented with weight decay regularization in these models but did not observe consistent or noteworthy improvements.

3.5.5 The Forward Pass

In the forward pass, the model processes the sampled batch of inputs $\mathbf{x}^{(i)}$ in parallel, where $i = 1, 2, \ldots, N$ represents the i-th sample in the batch. The model computes the logits $\mathbf{z}^{(i)}$ (i.e. the outputs of the last classification layer) by the end of a forward pass. A softmax function is then applied to normalize the logits and convert them into label probabilities $\hat{\mathbf{y}}^{(i)}$. Since isolated sign language recognition is a multi-class classification problem, we use the cross-entropy loss to measure the discrepancy between the model's predictions $\hat{\mathbf{y}}^{(i)}$ and the true labels $\mathbf{y}^{(i)}$:

$$L(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) = -\sum_{c=1}^{C} y_c^{(i)} \log(\hat{y}_c^{(i)}) \tag{3.10}$$

where

- C is the number of classes in the dataset
- $\mathbf{y}^{(i)} = [y_1^{(i)}, y_2^{(i)}, \ldots, y_C^{(i)}]$ is the one-hot encoded true label vector for the i-th sample.
- $\hat{\mathbf{y}}^{(i)} = [\hat{y}_1^{(i)}, \hat{y}_2^{(i)}, \ldots, \hat{y}_C^{(i)}]$ is the predicted probability vector for the i-th sample obtained from the softmax.

However, since the true labels $\mathbf{y}^{(i)}$ provided by the dataset (especially MS-ASL and WLASL) are not always correct due to automated label extraction, which is error-prone, we incorporate label smoothing [95] into the loss. The one hot encoding for the true labels $\mathbf{y}^{(i)}$

is adjusted by smoothing it with a constant α:

$$L_{\text{smooth}}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) = -\sum_{c=1}^{C} \left[(1-\alpha) \cdot y_c^{(i)} + \frac{\alpha}{C}\right] \log(\hat{y}_c^{(i)}) \tag{3.11}$$

The label smoothing hyperparameter α is used to prevent overconfidence in the given labels. We set $\alpha = 0.1$ as we found it to work well for the datasets. In our experiments, label smoothing has often improved the generalization accuracy by about 1%, as preventing overconfidence in noisy labels can be considered a form of regularization. The total loss corresponding to the samples in the batch is computed as the average of each sample's smooth loss:

$$L_{\text{batch}} = \frac{1}{N} \sum_{i=1}^{N} L_{\text{smooth}}(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) \tag{3.12}$$

3.5.6 The Backward Pass

In the backward pass, the gradients of the batch loss L_{batch} with respect to the model's weights θ are computed with backpropagation, as an application of the chain rule:

$$\frac{\partial L_{\text{batch}}}{\partial \theta} = \frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} \frac{\partial L_{\text{smooth}}^{(i)}}{\partial z_c^{(i)}} \cdot \frac{\partial z_c^{(i)}}{\partial \theta} \tag{3.13}$$

where the gradient of the smooth loss with respect to the logits $z_c^{(i)}$ is:

$$\frac{\partial L_{\text{smooth}}^{(i)}}{\partial z_c^{(i)}} = \hat{y}_c^{(i)} - \left[(1-\alpha) \cdot y_c^{(i)} + \frac{\alpha}{C}\right] \tag{3.14}$$

The process of computing $\frac{\partial z_c^{(i)}}{\partial \theta}$ is automated by Pytorch as the gradient of a layer's output with respect to its parameters is predefined. Finally, the model weights are updated with the computed gradients using the update rules of Adam optimization [73]. This weight update minimizes the loss and consequently improves the accuracy.

3.6 Strategies for Faster Training

Training 2D and 3D deep convolutional neural networks on a large number of videos for many epochs is very resource-intensive and time-consuming. In this section, we describe the most impactful strategies we employed to accelerate the training pipeline.

3.6.1 Training with Mixed Precision

Mixed precision training uses both half-precision (16-bit) for computationally intensive tasks, such as the forward and backward passes, and full single-precision (32-bit) for operations where high precision is required, such as weight updates. Empirical evidence has shown that mixed precision training can significantly speed up training without compromising the accuracy of various deep learning models [96]. Moreover, mixed precision training uses significantly less memory since 16-bit numbers use half the memory of 32-bit numbers.

This has enabled our training pipeline to double the batch size from 16 to 32 clips per batch in training I3D on AUTSL videos while consuming a similar amount of GPU memory, about 40GB. Switching from full-precision to mixed-precision has also reduced the training time per epoch from 27 minutes to 22 minutes. In this case, the training time per epoch is not reduced by half because 18 minutes of the epoch time are spent on non-GPU work (bottleneck) such as video loading, reading, and preprocessing. This will be further optimized as discussed in distributed parallel training (Section 3.6.2).

In mixed precision training, weight updates are done in full precision (FP32) because updating weights with small gradients in half-precision (FP16) can be imprecise (e.g., 1 + 0.0001 results in 1) [97]. The weights used in the forward and backward passes are FP16-casted versions of the master copy of the model's weights, which is maintained in FP32 and updated at each optimization step. To avoid the underflow of very small gradients computed in FP16 during the backward pass, the loss is dynamically scaled by a factor such as 1024, causing a corresponding scaling of the gradient. The gradients are then scaled back in FP32 before being applied in the weight update.

3.6.2 Distributed Parallel Training

Our training pipeline uses a single node with 2 or 3 GPUs for data parallelism. We implemented data parallelism in PyTorch using the DistributedDataParallel (DDP) module [98]. A separate process is spawned for each GPU, and the model is replicated across the GPUs. At the beginning of each epoch, the dataset is randomly shuffled and evenly split using DistributedSampler, with each split assigned to a different replica of the model (e.g., different GPU). Each GPU independently computes the forward and backward passes for its portion of the dataset. Afterward, the gradients from all replicas are synchronized to compute their average. The model's weights on all replicas are updated in the optimization step using the averaged gradients. To track the accuracy and loss for each epoch, we apply reduce-sum operations on the statistics computed by each process.

In addition to multiple GPUs, we use 12 CPU cores (workers) to speed up data loading and preprocessing. We assign (12/num of GPUs) CPU cores to each process. Since our epoch time is dominated by CPU data tasks, splitting the data across multiple processes (with one GPU for each process) has had the most significant impact, reducing the 22 minutes per epoch in mixed-precision single-GPU training by 45% to 12 minutes per epoch in mixed-precision two-GPU training.

Table 3.1 summarizes the number and types of GPUs we used for training the 3D convolutional models on sign datasets. I3D is the most efficient model in terms of training speed and memory. Although training I3D on MS-ASL or WL-ASL on a single 24-GB RTX 3090 GPU was possible memory-wise due to the smaller batch size, we used 2 GPUs to accelerate training. F3D is trained on a batch of 9 as it is divisible by the number of GPUs used in its training. Training R3D and F3D on the AUTSL dataset with larger crops and batch sizes was done on A100 GPUs due to the lack of access to more than 3 RTX 3090 GPUs.

Table 3.1: Summary of hyper-parameter tuning for training the RGB models (I3D and R3D) and the flow model (F3D) on the sign datasets. All listed models were trained with label smoothing=0.1, Adam optimization, and one-cycle learning rate schedule.

	MS-ASL / WL-ASL datasets			AUTSL dataset		
	I3D	R3D	F3D	I3D	R3D	F3D
Learning rate	1e-3	3e-4	1e-3	1e-3	3e-4	1e-3
Batch size	8	8	9	32	32	32
Dropout prob.	0.5	0.2	0.5	0.5	0.2	0.5
Training epochs	100	100	100	40	40	40
GPUs used	2 x RTX3090	2 x RTX3090	3 x RTX3090	2 x RTX3090	2 x A100	2 x A100
Training crop	224x224 from 256x256			256x256 from 512x512		
Inference crop	256x256 (full frame)			center 320x320 from 512x512		

3.7 Inference Pipeline for RGB and Flow Models

In this section, we describe the details of the inference pipeline we designed to evaluate the performance of the RGB and flow models on the validation and test splits of the datasets. To track the generalization performance of a model on the validation set during training, we perform an inference (test) epoch after the completion of each training epoch or after every five training epochs if the inference epoch is time-consuming. An inference epoch is often faster than a training epoch because there is no gradient computation (backward pass), and much fewer samples are assigned to the validation and test sets than the training set. Before every inference epoch, dropout regularization is disabled, and batch normalization layers of the model are fixed to use the same statistics computed during training.

3.7.1 Loading and Resizing Frames

To perform an inference on videos in the validation or test splits on MS-ASL and WL-ASL datasets, the video is first loaded by the video reader and resized to a fixed 256×256

resolution. The video is then normalized using the same mean and standard deviation in the training pipeline. We apply the model convolutionally over the entire video, both spatially and temporally, where the output features corresponding to each $64 \times 224 \times 224$ region in the input are averaged by the model's last pooling operation before its classification layer. This has shown consistent improvement over applying the model once on a 224×224 center crop.

We follow a similar process for performing inference on the AUTSL dataset. However, we do not use the full input resolution of 512×512 since that is very computationally costly. Instead, we apply the model convolutionally over a 320×320 center crop, covering 39% of the frame area. Using a 256×256 center crop instead of 320×320 negatively impacts the accuracy, as 75% of the input frame is lost. On the other hand, increasing the center crop to 384×384 did not improve the accuracy. We also experimented with reducing the original resolution to 384×384 and taking a 256×256 center crop, covering 44% of the frame area, but this approach underperformed the 320×320 center crop taken from 512×512. This highlights the importance of input resolution for effective sign recognition. We leave this to future work.

3.7.2 Accelerating Inference

Since the inference epoch runs in the same process as the training epoch, we can exploit the same data parallelism as the training pipeline by assigning a portion of the validation set to each GPU. However, we use a batch size of one in each process because all video frames are used during inference, and it is not feasible to form a mini-batch with multiple samples when they have varying numbers of frames.

To track the validation loss and accuracy, each process accumulates its own statistics, including the loss, number of correctly classified samples, and the total number of samples. After the inference epoch ends, a reduced-sum operation is applied to the accumulated variables to compute the total accuracy and loss for the entire validation set. Because the

number of samples in the validation set is not necessarily divisible by the number of GPUs used, the calculated accuracy will be slightly less accurate. After the training ends, we run the last checkpoint on the validation set on one GPU without data parallelism to calculate the precise accuracy.

3.8 Training and Inference for Skeleton Models

In this section, we describe the training and testing (inference) pipelines for the skeleton models (GRU, Transformer encoder, and ST-GCN) and highlight how they differ from the pipelines described in Sections 3.5 and 3.7. The process for training and testing skeleton models is generally similar to other models, unless stated otherwise.

Data Loading

The first stage for a skeleton model involves skeleton extraction from every frame in the input RGB video using MediaPipe [34] module. To save considerable time in training and testing, we pre-extract skeleton data from all frames in all videos of the sign language datasets and store it in a Python dictionary saved as a pickle file. Prior to training or testing, the pickle file is loaded into RAM once.

This approach is feasible because skeleton data has a relatively small size compared to other modalities (e.g. 1.25 GB for the skeleton of AUTSL training set). In contrast, storing the extracted optical flow for the same dataset at a resolution of 256×256 with full precision (32-bit float) would require approximately 880 GB. This substantial storage requirement is why we opted to extract optical flow on the fly for each video selected in a batch.

Data Augmentation

Similar to RGB and optical flow models, we randomly select 64 consecutive frames from the skeleton data of each video sampled in the batch during training. If a video has fewer than 64 frames, the remaining x frames are filled by looping the skeleton data of the first frame up to the x-th frame. No spatial cropping is applied as the skeleton extraction was

run on the full frames. We also experimented with other forms of augmentation, such as random horizontal flipping of keypoint coordinates, but that did not help the performance. For inference, we use the skeleton data extracted from all frames in the video.

Normalization

Since the horizontal and vertical coordinates of the keypoints extracted by MediaPipe are already normalized to [0, 1], there is no need to divide them by the width and height of the video frame, respectively. We experimented with normalizing the coordinates of all keypoints such that the nose coordinate in each frame is assigned as the origin $(0, 0)$, but this approach did not perform well. This is likely because many undetected keypoints in the skeleton are also assigned the same coordinate $(0, 0)$. Therefore, we use the default coordinate system, where the upper-left corner of the frame is the origin $(0, 0)$.

Training Hyperparameters

Similar to other models, all training hyperparameters are tuned based on the performance on the validation set. We use a batch size of 32 for the AUTSL dataset and 8 for the MSASL and WLASL datasets. A label smoothing of 0.1 is applied to the loss function, as described in Section 3.5.5. The weight decay used for regularization is set to 1×10^{-3} for AUTSL, MSASL-1000, and WLASL-2000, and 1×10^{-2} for MSASL-100 and WLASL-100. All skeleton models are trained for 200 epochs across all datasets.

We find a suitable learning rate using the process described in Section 3.5.4. The learning rate is set to 1×10^{-2} for AUTSL, and 3×10^{-3} for all other datasets. For skeleton models, we use the Reduce-on-Plateau learning rate schedule with SGD optimization (momentum = 0.9) as we found this combination to perform better than the One-Cycle schedule with Adam optimization used for RGB and optical flow models. In the Reduce-on-Plateau schedule, the learning rate is reduced by half if the validation loss does not decrease for 10 consecutive epochs.

Accelerating Training and Inference

Both mixed precision training and distributed training strategies are applicable to skeleton models. However, due to the significantly lower computational cost of the second stage of skeleton models (e.g. ST-GCN) compared to RGB and optical flow models, we did not need to use more than one GPU for training and testing skeleton models within a reasonable time. We apply mixed precision training as described in Section 3.6.1 to the skeleton models to further speed up training.

3.9 Stream Fusion Methods

In this section, we investigate two main fusion methods to combine models. In post-classification fusion, we combine the classification scores (e.g., logits or prediction probabilities) from each stream after their respective classification layers. In pre-classification fusion, we fuse the stream features prior to the classification layer. The methods and results for each fusion method are presented in the following subsections.

3.9.1 Post-Classification Fusion

The direct outputs of a deep learning model's classification layer are known as logits, which represent unnormalized log probabilities for each class. These scores can be converted into probabilities that sum to 1 by applying a softmax function. However, we choose to combine the raw logits rather than the softmax probabilities because softmax accentuates the difference between the top prediction and other close contenders. This can reduce the usefulness of the combination if the correct prediction ranks second, as the softmax transformation would downplay its relative score. Our experiments further support this choice, showing slightly better results when combining models' logits instead of probabilities. We define combining the logits as a weighted average, described formally as follows.

- Let $D_{\text{val}} = \{(x_i, y_i)\}_{i=1}^{N}$ be the validation dataset, where x_i is the input and y_i is the true label for sample i.

- Let logits1_i and logits2_i be the logits (pre-softmax scores) output by Model 1 and Model 2 for sample i, respectively.

- The weighted combination of the logits in a two-stream network for sample i and class c is defined as:

$$s_i^{(c)}(\alpha) = \alpha \cdot \text{logits1}_i^{(c)} + (1 - \alpha) \cdot \text{logits2}_i^{(c)} \tag{3.15}$$

where α represents the weight assigned to the outputs (logits) of Model 1, while Model 2's outputs are weighted by $1 - \alpha$.

A straightforward approach is to set $\alpha = 0.5$, effectively giving equal weight to both streams. However, since the models differ in accuracy, a simple average is often suboptimal, as assigning equal weight to the less accurate model may limit the overall improvement of the ensemble. Instead, we propose searching for the optimal weight α that maximizes the validation accuracy:

$$\alpha^* = \arg\max_{\alpha \in [0,1]} \sum_{i=1}^{N} \mathbb{I}\left(\arg\max_c s_i^{(c)}(\alpha) = y_i\right) \tag{3.16}$$

where $s_i^{(c)}(\alpha)$ is defined in Eq. 3.15, and $\mathbb{I}$ is the indicator function that equals 1 if the predicted class matches the true class y_i, and 0 otherwise. To find α^*, which maximizes the accuracy of the ensemble, we implement a grid search optimization strategy since accuracy is not a differentiable metric and, therefore, cannot be maximized directly using gradient-based optimization.

In the grid search, each α in the interval [0, 1] with a step size of 0.01 is used to compute the ensemble accuracy, and the value corresponding to the maximum accuracy is chosen. We consider this grid search as our baseline, as it directly maximizes accuracy. Table 3.2 shows the result of the grid search along with other proposed optimization methods for a two-stream network with GCN and I3D representing Model 1 and Model 2, respectively, on the MSASL1000 dataset. In this case, the grid search found that weighting the logits of GCN with 0.29 and the logits of I3D with 0.71 achieves a maximum validation accuracy of

69.72%. We then apply the optimal weighting, α^*, found on the validation set to report the test set accuracy.

Table 3.2: Results of post-classification optimization methods on MSASL-1000 dataset

Optimization Method	**Criteria**	**α^* (for GCN)**	**Validation Accuracy**	**Test Accuracy**
Grid Search (baseline)	Maximize Accuracy (Eq. 3.16)	0.29	69.72%	74.20%
Adam	Minimize Hinge Loss (Eq. 3.19)	0.3155	69.50%	74.30%
L-BFGS	Minimize Hinge Loss (Eq. 3.19)	0.3155	69.48%	74.30%
L-BFGS	Minimize CE Loss (Eq. 3.17)	0.5326	67.64%	73.51%
L-BFGS (Per-Class α)	Minimize CE Loss (Eq. 3.21)	1000 values	**71.46%**	72.37%
L-BFGS (Per-Group α)	Minimize Hinge Loss (Eq. 3.23)	[0.3146, 0.3086, 0.3212, 0.3019, 0.3005]	69.24%	**74.45%**

In addition to grid search, we explore gradient-based methods to find α^*. Instead of directly maximizing the accuracy, which is not differentiable, we minimize a differentiable loss function as a surrogate for accuracy, therefore allowing the use of backpropagation to learn α^*. All other weights of the models prior to the output of the classification layer are fixed, as this is a post-classification fusion. Two common choices for the surrogate loss functions are cross-entropy loss and hinge loss.

Cross-entropy loss is widely used for classification tasks because it encourages the model to output a high probability for the correct class, which directly improves prediction confidence. Using the cross-entropy loss, the optimization objective we propose can be restated as follows:

$$\alpha^* = \underset{\alpha \in [0,1]}{\arg\min}\ L_{\text{CE}}(\alpha) \tag{3.17}$$

where $\alpha = \sigma(\widetilde{\alpha})$ is the result of applying a sigmoid function to the unconstrained parameter $\widetilde{\alpha}$, mapping it to the [0,1] range. $L_{\text{CE}}(\alpha)$ denotes the cross-entropy loss function defined as:

$$L_{\text{CE}}(\alpha) = -\sum_{i=1}^{N} \log\left(\text{Softmax}\left(s_i(\alpha)\right)_{y_i}\right) \tag{3.18}$$

Alternatively, hinge loss, which is typically used in support vector machines, ensures that the score for the correct class is separated from the incorrect classes by a margin. Minimizing the multi-class hinge loss over α is mathematically expressed as follows:

$$\alpha^* = \underset{\alpha\in[0,1]}{\arg\min}\ L_{\text{hinge}}(\alpha) \tag{3.19}$$

$$L_{\text{hinge}}(\alpha) = \sum_{i=1}^{N}\sum_{j\neq y_i} \max\left(0, 1 - \left[s_i^{(y_i)}(\alpha) - s_i^{(j)}(\alpha)\right]\right) \tag{3.20}$$

where $\alpha = \sigma(\widetilde{\alpha})$, and $s_i^{(c)}(\alpha)$ as defined in Eq. 3.15.

In the single-weight optimization methods, learning α^* by minimizing the cross-entropy loss produces lower validation accuracy compared to learning α^* by minimizing the hinge loss (Table 3.2, rows 3 and 4). This is because hinge loss focuses on the scores most close to the correct class, encouraging adjustment that improves the margin between them. In contrast, cross-entropy loss is influenced by the predicted probability of all classes. Therefore, in datasets with many classes (e.g., MSASL1000), the gradient signal can become less focused as the optimization adjusts α in ways that slightly improve probabilities across many classes but do not directly help separate the correct class from the most confusable ones.

For the gradient-based optimization, first-order methods such as SGD or Adam can be used to update α. However, second-order optimization methods often provide faster convergence to the optimal value in fewer steps. We chose the L-BFGS algorithm, a quasi-Newton method that approximates second-order optimization without computing the full Hessian matrix [99]. Other second-order methods, such as conjugate gradient, could also be

applied, but we opted for L-BFGS due to its accessible implementation in PyTorch. Table 3.2 (rows 2 and 3) shows that both Adam and L-BFGS converge to approximately the same α^*.

In the single-weight optimization methods discussed so far, all classes are weighted with the same value, α^*. If the hypothesis that an RGB stream or a skeleton steam is particularly useful for recognizing certain classes of signs, then we should expect accuracy to improve by adopting multiple α's that weight classes differently per stream. To evaluate this hypothesis, we modify α from Eq 3.15 to be a vector of C values (weights), where C is the number of classes. For this approach, we use cross-entropy loss because the C weights should be influenced by the probabilities of all classes, not just the margin between the correct class and its close contenders. The optimization objective we propose is therefore restated as follows:

$$\boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha} \in [0,1]^C}{\arg\min}\ L_{\text{CE}}(\boldsymbol{\alpha}) \tag{3.21}$$

where

- $\boldsymbol{\alpha} = \text{Softmax}(\widetilde{\alpha}_1, \widetilde{\alpha}_2, \ldots, \widetilde{\alpha}_C) = (\alpha_1, \alpha_2, \ldots, \alpha_C)$ representing the vector of fusion weights, where each α_c corresponds to a specific class c. The Softmax function is applied to the unconstrained weights $\widetilde{\alpha}_c$, so that:

$$\alpha_c \in [0, 1] \quad \text{and} \quad \sum_{c=1}^{C} \alpha_c = 1, \quad \text{for } c = 1, 2, \ldots, C.$$

- $L_{\text{CE}}(\boldsymbol{\alpha})$ is the cross-entropy loss, similar to that in Eq 3.18, but as a function of a weight vector $\boldsymbol{\alpha}$ rather than a single weight α.

-

$$s_i^{(c)}(\boldsymbol{\alpha}) = \alpha_c \cdot \text{logits1}_i^{(c)} + (1 - \alpha_c) \cdot \text{logits2}_i^{(c)} \tag{3.22}$$

The validation and test accuracies of this approach are reported in Table 3.2 (5th row). This is a clear case of overfitting, where fitting 1000 fusion weights resulted in higher validation accuracy than all other single-weight methods but worse performance on the test

set. One reason for this outcome is that the majority of the 1000 classes in MSASL dataset have a low number of validation samples relative to the number of parameters the optimization needs to learn. Moreover, the distribution of the validation set in MSASL-1000 differs slightly from that of the test set, making the overfitted weights less optimal for the test set.

To mitigate overfitting to the validation set, we explore reducing the number of fusion weights by learning a weight per group of classes rather than a weight per class. In this approach, classes are mapped into G groups, where G is significantly smaller than C (i.e. $G << C$). The G fusion weights are then learned by optimizing the following objective:

$$\boldsymbol{\alpha}^* = \underset{\boldsymbol{\alpha} \in [0,1]^G}{\arg\min}\ L_{\text{hinge}}(\boldsymbol{\alpha}) \tag{3.23}$$

where

- $\boldsymbol{\alpha} = \text{Softmax}(\widetilde{\alpha}_1, \widetilde{\alpha}_2, \ldots, \widetilde{\alpha}_G) = (\alpha_1, \alpha_2, \ldots, \alpha_G)$ representing the vector of fusion weights, where G is the number of groups and $g(c)$ maps each class c to its corresponding group g. The Softmax function is applied to the unconstrained weights $\widetilde{\alpha}_{g(c)}$, so that:
$$\alpha_{g(c)} \in [0,1] \quad \text{and} \quad \sum_{c=1}^{C} \alpha_{g(c)} = 1, \quad \text{for } c = 1, 2, \ldots, C.$$
- $L_{\text{hinge}}(\boldsymbol{\alpha})$ is the hinge loss, similar to that in Eq 3.20, but as a function of a weight vector $\boldsymbol{\alpha}$ rather than a single weight α.
-
$$s_i^{(c)}(\boldsymbol{\alpha}) = \alpha_{g(c)} \cdot \text{logits1}_i^{(c)} + (1 - \alpha_{g(c)}) \cdot \text{logits2}_i^{(c)} \tag{3.24}$$

It is worth noting that Eq. 3.24 represents the most general formulation of weighted combination of logits among our proposed methods. By setting $g(c) = c$ (each class has its own group), Eq. 3.24 simplifies to the per-class weight fusion in Eq. 3.22. By setting $g(c) = 1$ (all classes belong to a single group), Eq. 3.24 reduces to the single-weight fusion in Eq. 3.15.

The result of this approach is reported in Table 3.2 (last row). In this setup, $g(c)$ represents a mapping in which the 1000 classes are grouped into five bins based on the number of training samples per class in the training set: [0, 10), [10, 20), [20, 30) , [30, 40), and [40, 50). While learning a weight per group according to this mapping avoids the overfitting issue observed with the per-class weight method, it does not significantly improve validation or test accuracy over single-weight methods. Furthermore, the five learned α values of this approach are nearly the same (around 0.3), indicating that the contribution of the skeleton stream (GCN) does not vary by much as a function of the number of training samples in a class.

Overall, the optimization methods in Eq. 3.21 and Eq. 3.23 provide no evidence supporting the hypothesis that the RGB stream (I3D) or the skeleton steam (GCN) is particularly useful for recognizing certain classes of signs over the other. Therefore, the null hypothesis cannot be rejected. Alternative group mappings $g(c)$, such as clustering classes based on feature similarities or learning $g(c)$ directly from the data, may provide better results. We leave exploring this option as a future work.

Generalizing to Multiple Streams

Let logits1_i, logits2_i, and logits3_i be the logits (pre-softmax scores) output by Model 1, Model 2, and Model 3 for sample i, respectively. The weighted combination of the logits in a three-stream network for sample i and class c is defined as:

$$s_i^{(c)}(\alpha_1, \alpha_2) = \alpha_1 \cdot \text{logits1}_i^{(c)} + \alpha_2 \cdot \text{logits2}_i^{(c)} + (1 - \alpha_1 - \alpha_2) \cdot \text{logits3}_i^{(c)} \tag{3.25}$$

where α_1, α_2, and $(1 - \alpha_1 - \alpha_2)$ represent the post-classification fusion weights assigned to the outputs (logits) of Model 1, Model 2, and Model 3, respectively. The weights satisfy $0 \leq \alpha_1, \alpha_2 \leq 1$ and $\alpha_1 + \alpha_2 \leq 1$. For N-stream networks, this formulation can be generalized to N fusion weights $\alpha_1, \alpha_2, \ldots, \alpha_{N-1}$, with the final weight computed as $1 - \sum_{j=1}^{N-1} \alpha_j$, to ensure their total equals 1.

To determine the best fusion weights in three-stream networks, we perform a grid search over the interval $[0, 1]$ for both α_1 and α_2 with a step size of $\delta = 0.01$. The optimal weights (α_1^*, α_2^*) are those that maximize the top-1 accuracy of the ensemble on the validation set. The detailed procedure is outlined in Algorithm 1.

Algorithm 1 Grid search for optimal fusion weights in a three-stream network

Input: Validation dataset $D_{\text{val}} = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i is the input and y_i is the true label for sample i; Step size $\delta = 0.01$ for grid search

Output: Optimal weights (α_1^*, α_2^*) that maximize top-1 accuracy on the validation set

Initialize $\alpha_1^* = 0$, $\alpha_2^* = 0$, and max_accuracy $= 0$
for each α_1 in $[0, 1]$ with step size δ **do**
  for each α_2 in $[0, 1 - \alpha_1]$ with step size δ **do**
    Compute the fusion weight for the third model as $\alpha_3 = 1 - \alpha_1 - \alpha_2$
    Compute combined logits for each sample (x_i, y_i) in D_{val}:

$$s_i^{(c)}(\alpha_1, \alpha_2) = \alpha_1 \cdot \text{logits1}_i^{(c)} + \alpha_2 \cdot \text{logits2}_i^{(c)} + \alpha_3 \cdot \text{logits3}_i^{(c)}$$

    Calculate top-1 accuracy on D_{val}:

$$\text{accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(\arg\max_c \, s_i^{(c)}(\alpha_1, \alpha_2) = y_i \right)$$

    if accuracy $>$ max_accuracy **then**
      Set max_accuracy $=$ accuracy
      Set $\alpha_1^* = \alpha_1$ and $\alpha_2^* = \alpha_2$
    end if
  end for
end for
return Optimal weights (α_1^*, α_2^*)

3.9.2 Pre-Classification Fusion

In addition to post-classification fusion, which is represented as a linear function (weighted average) of the steam outputs (Eq. 3.15), we explore a non-linear fusion of the stream features prior to the classification layer. This approach allows us to evaluate whether learning complex, non-linear interactions between the stream features can improve the performance

of the two-stream network. Figure 3.13 illustrates our proposed pre-classification fusion of GCN and I3D features.

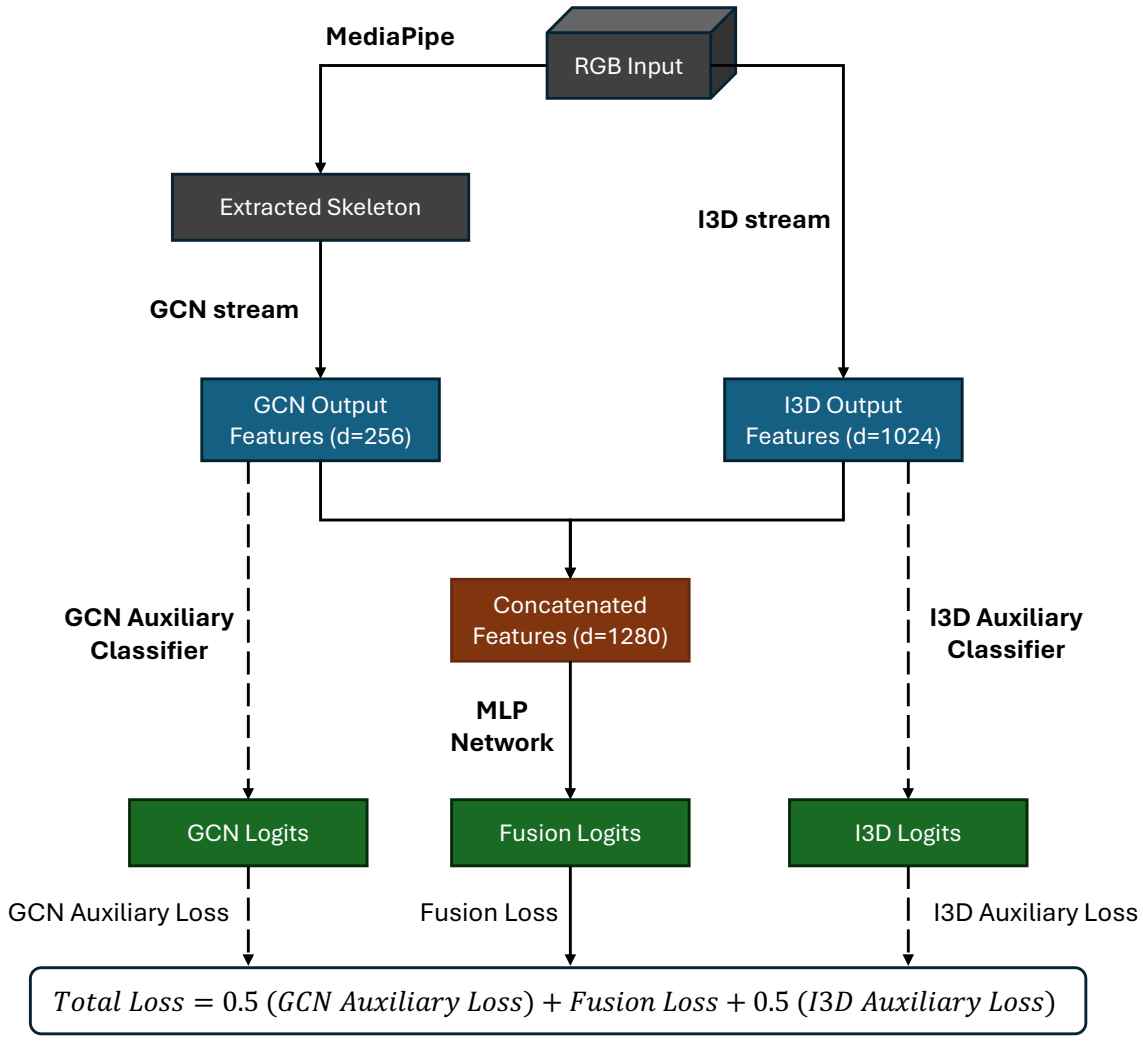


Figure 3.13: Proposed pre-classification fusion method. Output features of the two streams are concatenated before the classification layer, then fed into MLP to learn non-linear feature fusion.

The "output features" or simply "features" refers to the output of the penultimate layer in the I3D and GCN streams, which is the last layer in the stream before the classification layer. The output features of the two streams are concatenated to form a 1280-dimensional feature vector. This concatenated feature vector is then passed through a multi-layer per-

ceptron (MLP) consisting of two fully connected layers, each with 512 neurons and ReLU activations. Finally, a fully connected classification layer is applied to obtain the fusion logits corresponding to the concatenated features.

To ensure that the learned fusion does not rely heavily on the features of a single stream, we apply auxiliary classifiers to the features of each stream to minimize the loss of each stream in addition to the fusion loss. The three losses are combined as follows:

$$\text{Total Loss} = \text{Fusion Loss} + 0.5 \times (\text{GCN Auxiliary Loss}) + 0.5 \times (\text{I3D Auxiliary Loss}) \quad (3.26)$$

Including the auxiliary loss of each stream in the total loss encourages the two-stream network to learn output features that are useful on their own, while the inclusion of the fusion loss ensures that the learned features are also useful when combined. This design has a regularization effect as it constrains the feature space to those features that are beneficial both individually and when fused.

The logits obtained from the auxiliary classifiers can also be utilized for post-classification fusion to learn a weighted combination of the GCN, I3D, and fusion logits as described in Algorithm 1. This approach effectively combines both pre-classification and post-classification fusion methods.

The fused network is fine-tuned end-to-end, with the I3D and GCN streams initialized from the coarse-to-fine pretrained models. For example, if the fused model is to be fine-tuned on MSASL1000 dataset, the I3D and GCN streams are initialized with weights pretrained sequentially on Kinetics, AUTSL, and then MSASL1000.

Ablation Study

Table 3.3 shows the effect of removing various training aspects of the fused model on the MSASL-1000 dataset. Including all components achieves 67.22% validation accuracy based on the fusion logits alone and about 69% accuracy when using a weighted ensemble of the three logits. Training the fused model without the auxiliary classifiers and losses reduces the

validation accuracy to 64.24%, and freezing the weights of the two streams during finetuning further drops the accuracy to 62.04%.

Table 3.3: Ablation study on different training aspects of the pre-classification fusion method.

Unfrozen I3D and GCN	Auxiliary classifiers	Pretrained on MS1000	Finetuning epochs	Accuracy of Fusion Head		Accuracy of Ensemble (3 heads)	
				Val.	Test	Val.	Test
✗	N/A	✓	20	62.04	68.11	–	–
✓	✗	✓	20	64.24	68.68	–	–
✓	✓	✗	100	64.96	69.84	69.59	74.06
✓	✓	✓	20	**67.22**	71.96	68.98	73.87
✓	✓	✓	40	67.18	**72.54**	**69.77**	**74.49**

In addition to initializing the MLP weights randomly, we also experimented with pre-training the MLP on AUTSL using I3D and GCN with AUTSL-pretrained weights. The fused model was then trained end-on-end on the MSASL-1000 dataset for 100 epochs (Table 3.3, third row). Despite being trained for more epochs, this approach does not improve the ensemble accuracy over the method where I3D and GCN are initialized with MSASL-1000 weights and trained for only 40 epochs. This demonstrates that it is better to train the two streams independently on the target dataset first, and then fine-tune them together on the target dataset.

The best approach in Table 3.3 did not improve the accuracy by a meaningful margin over the post-classification fusion baseline in Section 3.9.1 - Table 3.2 (69.77% vs 69.72%). This suggests that the linear combination of stream logits with fusion weight α learned through grid search is a strong baseline, and that learning a non-linear fusion of the stream features prior to the classification layer adds more complexity without substantial performance gain. Given these findings, we will only use post-classification fusion with grid search for the results of two- and three-stream networks discussed later in Chapter 4.

Chapter 4

Results and Analysis

This chapter is organized as follows. Section 4.1 defines the evaluation metrics used to report the performance of the models. Section 4.2 evaluates the impact of 3D convolution (slow fusion) compared to 2D convolution with late fusion of temporal features. Section 4.3 examines the impact of using optical flow compared to RGB as inputs to the 3D convolutional model and analyzes the effect of optical flow components. Section 4.4 presents the results and evaluates the performance of the skeleton models. Section 4.5 analyzes the impact of coarse-to-fine transfer learning and presents the results of all models after its application. Sections 4.6 and 4.7 present and analyze the results of all possible combinations of models in two- and three-stream networks, respectively. This chapter concludes by comparing our approach and results with state-of-the-art methods in Section 4.8.

4.1 Evaluation Metrics

Models are evaluated on the validation and test sets using accuracy. For multi-class classification problems, accuracy is the ratio of correct predictions to the total number of predictions:

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{Total predictions}} = \frac{\sum_{i=1}^{C} \text{True Positives}_i}{\sum_{i=1}^{C} \text{Total}_i} \tag{4.1}$$

where C is the number of classes in the dataset. This is also equivalent to the ratio of total number of true positives for each class, to the total number of instances across all classes.

A significant limitation of using accuracy alone as an evaluation metric is its potential to overestimate the performance of a classifier for imbalanced datasets, such as MS-ASL or WL-ASL, where the earlier classes (in terms of class id) have more samples than later ones. Simple accuracy gives equal weight to every instance in the dataset without considering the model's performance per class. Therefore, a model can perform well in recognizing common

signs (with many samples) while performing poorly on uncommon signs (with fewer samples) yet still achieving high accuracy due to the disproportionate representation of the majority classes.

While accuracy weights a model's performance on each instance equally, balanced accuracy is another metric that ensures the model's performance on each class is weighted equally, regardless of the class size. Rather than calculating the true positive rate over all instances, it calculates the true positive rate for each class individually and then averages them. This way, the model's performance on uncommon signs (minority classes) is considered as important as its performance on common signs.

$$\text{Balanced Accuracy} = \frac{1}{C}\sum_{i=1}^{C}\frac{\text{True Positives}_i}{\text{Total}_i} \tag{4.2}$$

Balanced accuracy is also equivalent to the average recall of each class, using macro averaging formula:

$$\text{Balanced Accuracy} = \frac{1}{C}\sum_{i=1}^{C}\text{Recall}_i = \frac{1}{C}\sum_{i=1}^{C}\frac{\text{True Positives}_i}{\text{True Positives}_i + \text{False Negatives}_i} \tag{4.3}$$

For the results discussed in this chapter, balanced accuracy will be referred to as "accuracy per-class" while standard accuracy will be referred to as "accuracy per-instance", "top-1 accuracy", or simply "accuracy".

Another useful evaluation metric is the top-5 accuracy, which follows the same formula as Eq. 4.1, except that an instance is considered correctly predicted (i.e counted as true positive) if its true label appears within the top 5 most likely predictions by the model. A strong performance on top-5 accuracy indicates the model can capture useful patterns in the data that allow the label to appear in the top-ranked predictions, even if it's not the highest probability (top-1).

Many of the results in this chapter include the Improvement Over Baseline (abbreviated as I.O.B.) metric to facilitate comparisons between the top-1 accuracies of various models

with respect to the baseline performance. We define this metric as follows:

$$\text{I.O.B. of the model} = \text{Accuracy of the model} - \text{Accuracy of the baseline model} \quad (4.4)$$

In our evaluation, the significance of a small improvement (e.g., I.O.B. = 1 percentage point) is not solely determined by the model's performance on a single dataset. However, if the model shows a consistent improvement, even a small one, across multiple datasets, our confidence in the significance of that result increases. This also eliminates the need for repeated runs on the same dataset, given that consistent improvement across different datasets already makes a stronger case. Moreover, the sign datasets we use (AUTSL, MSASL1000, and WLASL2000) have relatively large test set sizes (3,000 – 4,000 videos), resulting in stable results with low variance when training and testing are repeated on the same dataset.

4.2 The Impact of Slow Fusion (3D ConvNet)

A key research question of this dissertation is to determine whether using slow fusion—as represented by 3D convolutional networks—over 2D convolutional networks with late fusion is justified, given the added computational cost of 3D ConvNets. Since the difference between the two networks was not significant on action recognition datasets, as discussed in Chapter 2, we hypothesized that sign language video datasets are a better testbed for learning spatiotemporal features than general action recognition because learning to recognize signs often requires integrating information across frames. If this is true, then a 3D ConvNet must show a significant difference in performance from its 2D counterpart since it integrates spatiotemporal features across all layers rather than only the last.

To test this hypothesis, we compare the classification accuracy of the following four models, defined in Section 3.2.1.

1. 18-layer 2D ResNet with average pooling of features across frames (R2D18).

2. 18-layer 2D ResNet with GRU to integrate features across frames from the last layer before pooling (R2D18+GRU).

3. 50-layer 2D ResNet with average pooling of features across frames (R2D50).

4. 18-layer 3D ResNet with average pooling of temporal features (R3D18).

To ensure this is a fair comparison, we restricted it to one type of architecture (Residual Networks), which achieved state-of-the-art performance on the ILSVR 2015 challenge [45]. Moreover, we pre-train the baseline (R2D18) on the Kinetics400 dataset, the same dataset that R2D50 and R3D18 were trained on. Training on Kinetics400 is very time-consuming since it has an order of magnitude more training videos than AUTSL. In our experiment, training R2D18 for 50 epochs on Kinetics400 lasted for about 7 days, achieving a test accuracy of 66.4% which is 4 percentage points more than the accuracy reported for RGB stream in [2], demonstrating a better performance of our training pipeline on large scale action recognition dataset.

The optimal learning rate was tuned for each model independently using the method for finding the learning rate as discussed in Section 3.5.4. All models were trained for the same number of epochs; 40 epochs for AUTSL and 100 epochs for MSASL and WLASL datasets, using Adam optimization with the one-cycle schedule.

Tables 4.1, 4.2, and 4.3 show the validation and test accuracies of the four models on all sign datasets. The results demonstrate that R3D18 consistently outperforms its 2D counterpart (R2D18) in the top-1 accuracy by a significant margin of $7.9 - 19$ percentage points across different sign datasets. Other 2D Resnet models show improvements over the baseline, but to a lesser extent than R3D, when appending a recurrent net (R2D18 + GRU) or when increasing its depth (R2D50). This indicates that R2D18, even when pretrained on the same dataset as other models, is not sufficient enough to capture the subtleties of spatial and temporal features required for sign language datasets.

Table 4.1: Results of ResNet 2D and 3D models on AUTSL dataset. I.O.B. denotes improvement over baseline (R2D18).

Model	Validation		Test	
	Top-1	I.O.B.	Top-1	I.O.B.
R2D18 (baseline)	86.3	0	86.99	0
R2D18 + GRU	90.11	+3.81	88.80	+1.81
R2D50	89.7	+3.4	89.10	+2.11
R3D18	**94.16**	+7.86	**95.38**	+8.39

Table 4.2: Results of ResNet 2D and 3D models on MSASL-100 and MSASL-1000 dataset

Model	MSASL-100				MSASL-1000			
	Validation		Test		Validation		Test	
	Top-1	I.O.B.	Top-1	I.O.B.	Top-1	I.O.B.	Top-1	I.O.B.
R2D18 (baseline)	73.24	0	71.68	0	39.73	0	42.5	0
R2D18 + GRU	74.83	+1.59	73.94	+2.26	43.07	+3.34	43.03	+0.53
R2D50	76.51	+3.27	78.46	+6.78	48.29	+8.56	52.31	+9.81
R3D18	**82.16**	+8.92	**83.78**	+12.1	**57.24**	+17.51	**61.04**	+18.54

Table 4.3: Results of ResNet 2D and 3D models on WLASL-100 and WLASL-2000 dataset

Model	WLASL-100				WLASL-2000			
	Validation		Test		Validation		Test	
	Top-1	I.O.B.	Top-1	I.O.B.	Top-1	I.O.B.	Top-1	I.O.B.
R2D18 (baseline)	67.46	0	64.73	0	30.57	0	29.81	0
R2D18 + GRU	71.3	+3.84	66.67	+1.94	40.02	+9.45	38.53	+8.72
R2D50	72.19	+4.73	69.38	+4.65	40.53	+9.96	38.67	+8.86
R3D18	**80.18**	+12.72	**79.07**	+14.34	**49.57**	+19	**47.78**	+17.97

Appending a recurrent layer to the R2D18 model (R2D18+GRU) has improved the performance considerably on AUTSL and WLASL by a margin of $3-9$ percentage points similar to that of increasing its depth to 50 layers (R2D50). For MSASL, however, increasing the network depth helps more than appending a recurrent net. This is likely due to the variety of realistic backgrounds in MSASL, which are lacking in the two other datasets, making a deeper and more accurate image classifier such as R2D50 more important than temporal integration (R2D18+GRU).

Moreover, the results show that adding a temporal dimension across all layers (R3D18) consistently and significantly improves upon the baseline more than increasing the baseline's depth from 18 to 50 layers. A shallower 18-layer 3D ResNet outperforms the deeper 50-layer 2D ResNet by a margin of $4.5-8.9$ percentage points across the sign datasets. This highlights the efficiency of R3D18, as it can capture both spatial and temporal information with significantly fewer layers. As the classification problem becomes more challenging in datasets with a larger number of classes (such as MS1000 and WL2000), R3D demonstrates even more pronounced gains in accuracy over 2D convolutional models. This indicates that as the complexity of the signs grows, it becomes increasingly important to incorporate slow fusion to capture intricate discriminative features embedded within motion patterns.

Quantitative and Qualitative Analysis

The results in Tables 4.1, 4.2, and 4.3 demonstrate the impact of slow fusion on the overall (per-instance) accuracy. To analyze its impact on individual classes, Figure 4.1 shows a bar chart illustrating the difference in accuracy between R3D18 and R2D18 for each class in AUTSL dataset. This is calculated by subtracting R2D accuracy from R3D accuracy for each class. R3D improves the accuracy for the majority of classes (positive bars) while occasionally underperforming R2D (negative bars). The top three significant improvements have been observed in classes 16 (Ataturk), 47 (minute), and 95 (needle), with a 50% or more in accuracy difference and a p-value of less than 0.05 using the exact McNemar's test (i.e. Binomial exact test).

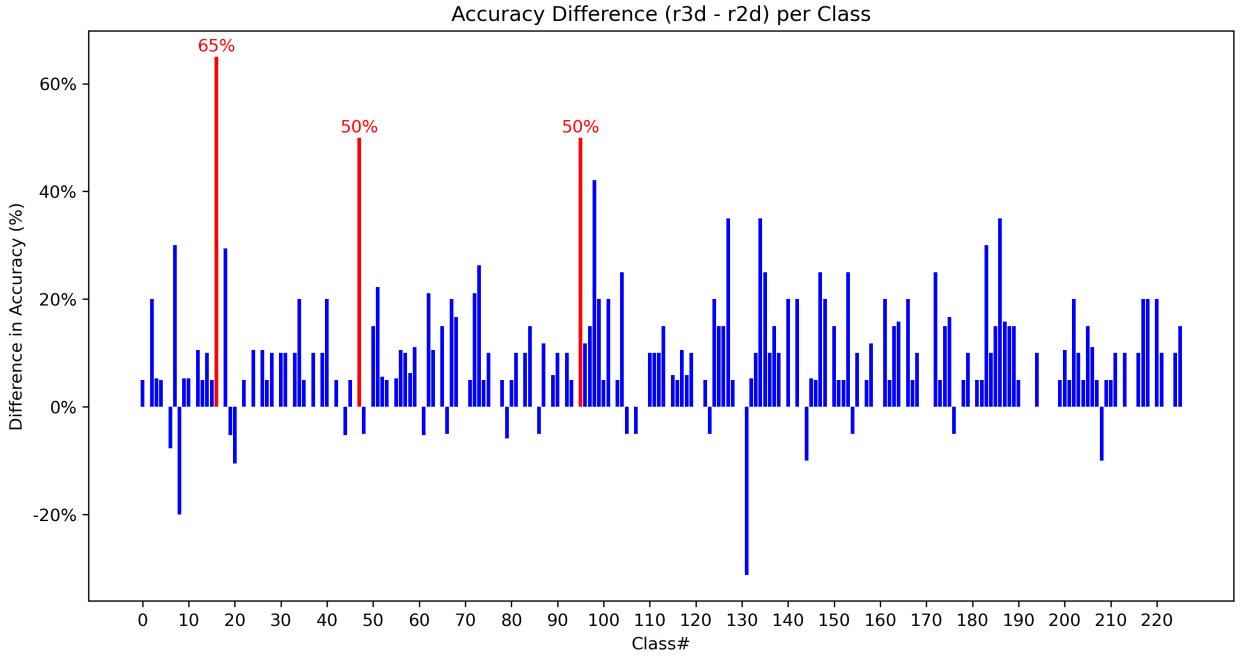


Figure 4.1: Accuracy difference (R3D - R2D) per class on AUTSL validation set

Table 4.4 shows details on the classification of these outliers, including the signs they were confused with by R2D and R2D+GRU models, causing a high error rate.

Table 4.4: Class confusion data with error rates for different models (R2D18, R2D18+GRU, R3D18), along with the p-values for statistical comparison.

True class #	**Confused with class**	**Class error rate**			**p-value**
		R2D18	**R2D18+GRU**	**R3D18**	
16 (Ataturk)	156 (potato), 172 (sugar)	70%	50%	5%	0.00098
47 (minute)	52 (doctor)	50%	40%	0%	0.00195
95 (needle)	134 (manager)	50%	41.67%	0%	0.03125
131 (napkin)	89 (gift)	6.25%	6.25%	37.5%	0.0625

R2D18 misclassified 70% of the signs in class #16, "Ataturk", often confusing them with the signs for "potato" (class #156) and sometimes for "sugar" (class #172). These three signs share similar spatial appearances but differ in their motions. As shown in Figure 4.2, class #16 is performed by placing the index finger on the face for about a second. Class #156 is performed similarly, but the hand rotates back and forth while the index finger is

placed on the face. In this case, the motion dynamics are more distinctive for these classes than appearance features.

Another example is signs of class #47, "minute," where R2D18 misclassified half of them as class #52, "doctor". The word "minute" is signed by tapping on the wrist (where a watch is typically placed) multiple times with the thumb and index fingers of the other hand touching each other (Figure 4.3). The sign for "doctor" is similar but involves either a single tap on the wrist in one variant or holding the wrist in another variant. Distinguishing between single and multiple taps with very similar spatial features is shown to be difficult again for R2D18 due to its lack of learning discriminative motion dynamics.

R2D+GRU slightly reduces the error rate for classes 16, 47, and 95 compared to R2D. As shown in Table 4.4, the error rate for each of these classes decreases as the temporal fusion method of features across frames becomes more sophisticated, progressing from simple late averaging in R2D to late fusion with R2D+GRU to slow fusion with R3D, where the error rate approaches 0. A notable exception (outlier) is class #131, "napkin", where R3D performed worse than R2D and R2D+GRU, although with a smaller difference in accuracy. In this case, 'napkin' was misclassified as 'gift' (class# 89). The primary distinction between these two signs is based on appearance rather than motion, as both start with a similar gesture near the nose but end with a throwing action for "napkin" and a holding gesture for "gift". This misclassification may result from R3D overemphasizing a narrow spatiotemporal region, potentially downplaying broader visual cues. However, since in this case the p-value of 0.0625 is greater than the significance level of 0.05, the null hypothesis that there is no significant difference in performance between R3D and R2D on class #131 cannot be rejected.

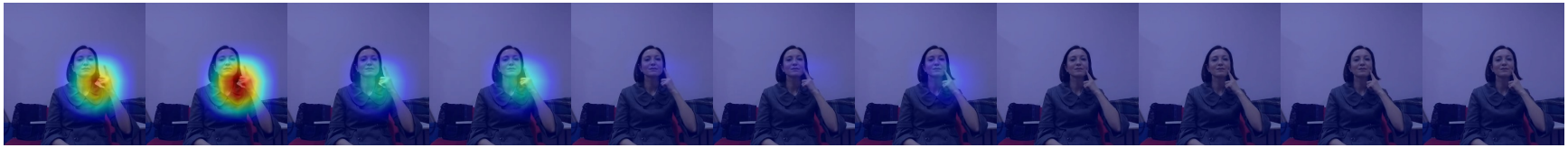
Comparing the class activation maps generated by each model can provide a deeper insight into how R3D outperforms R2D, particularly in classes such as 16 and 47, where motion plays a key role. Class Activation Mapping (CAM) [100] is a deep learning technique used to visualize the regions of the input that contribute most strongly to predicting a certain class. It achieves this by applying the class's learned weighted combination of activation maps

from the last convolutional layer. Gradient-weighted CAM (Grad-CAM) [101] extends CAM by using the gradient of the target class with respect to the feature maps of a convolutional layer. These gradients are averaged for each feature map and used as weights to combine the feature maps. To visualize the resulting weighted map for each frame, we resize it to the original input resolution using trilinear interpolation, then normalize it and overlay it on the input frame as a heatmap (Figures 4.2 and 4.3).

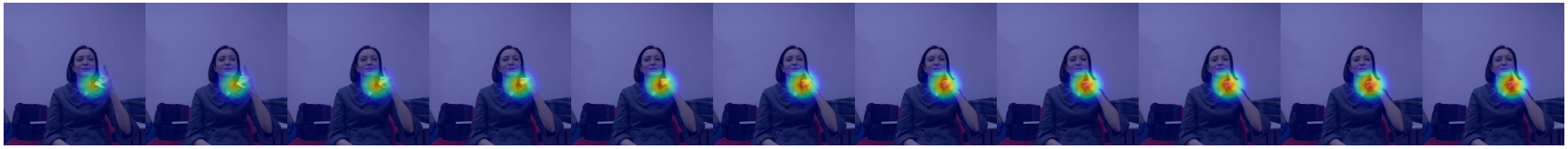
The last two rows in Figures 4.2 and 4.3 compare the heatmaps for R2D and R3D, illustrating the most influential regions in the input frames for classes 16 and 47, respectively. The heatmaps for R3D often demonstrate a narrower, more focused, and continuous spatiotemporal region, while R2D tends to highlight broader spatial regions in selected frames. In R2D, each frame is processed independently, so the frames that produce activation maps closer to maximizing the class score will contribute the most to the classification decision. R2D misclassified samples from class 16 as 156 or 47 as 52 because there were more frames producing activation maps aligned with the false class than the true class. This is especially likely in classes with similar visual cues. In contrast, classification in R3D is based on the detection of the sign pattern across multiple consecutive frames within a concentrated region of the input, which significantly improves differentiating between subtle motion cues, such as single vs. multiple taps or a stationary vs. rotating hand.

Sign for "Ataturk" (class #16) - classified correctly by R3D but missclassified by R2D as "potato"

Sign for "potato" (class #156)



R2D Grad-CAM for "Ataturk" sign

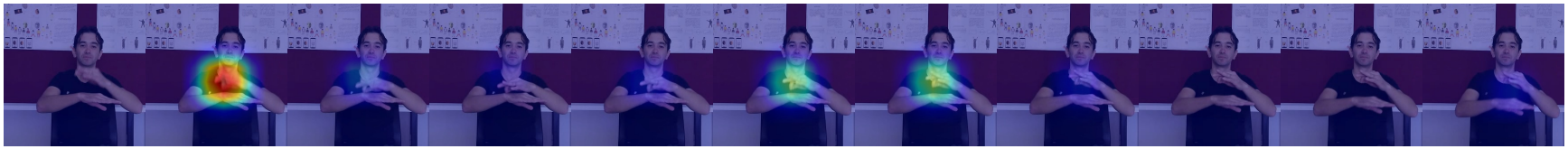


R3D Grad-CAM for "Ataturk" sign

Figure 4.2: Example of motion-sensitive sign "Ataturk" classified correctly by R3D and incorrectly by R2D and R2D+GRU due to motion being a key discriminative feature. The Gradient-weighted CAM (Grad-CAM) for "Ataturk" sign is visualized in the last two rows as heatmaps overlaid on the frames.

Sign for "minute" (class #47) - classified correctly by R3D but missclassified by R2D as "doctor"

Sign for "doctor" (class #52)



R2D Grad-CAM for "minute" sign

R3D Grad-CAM for "minute" sign

Figure 4.3: Example of motion-sensitive sign "minute" classified correctly by R3D and incorrectly by R2D and R2D+GRU due to motion being a key discriminative feature. The Gradient-weighted CAM (Grad-CAM) for "minute" sign is visualized in the last two rows as heatmaps overlaid on the frames.

4.3 The Impact of Optical Flow

In Section 4.2, it has been shown that a 3D convolutional network such as R3D is effective at capturing both spatial and temporal features of sign words. This raises the question of whether using an additional stream to explicitly learn motion patterns through optical flow is particularly beneficial for sign recognition, given the added computational overhead of extracting optical flow from RGB, especially in real-time applications.

Our hypothesis is when training a sufficiently deep 3D convolutional network on optical flow video input, the learning of motion patterns useful for sign recognition is primarily attributed to the slow fusion aspect of the network, rather than the optical flow input itself. This is significant because it suggests that multi-stream networks for sign recognition can achieve relatively high accuracy without relying on optical flow, therefore improving the computational efficiency of the inference pipeline and GPU memory requirement. We investigate this hypothesis in two ways.

1. Examining whether certain types of signs (classes), with very discriminative motion patterns, are significantly better classified using a 3D ConvNet with optical flow input than a 3D ConvNet with RGB input.

2. Analyzing the impact on the classification accuracy when key motion information is removed from the optical flow input (ablation study).

We choose the inception architecture as the backbone of the 3D convolutional network used for training on optical flow, rather than ResNet (R3D). We made this choice due to the availability of I3D pre-trained on optical flow of the Kinetics dataset [2], which allows for a fair comparison of performance with 3D convolutional networks pre-trained on RGB input from the same dataset. To distinguish between I3D network with RGB input and I3D with optical flow input, we refer to the latter as F3D, and retain the I3D name for the RGB-based model.

4.3.1 Performance of F3D and I3D

We use I3D as a baseline for comparing the performance of F3D because both models have similar backbone architecture (inception), but differ in input modality (RGB vs optical flow). In contrast, R3D differs from F3D in both architecture and input modality, making it harder to identify whether differences in performance are due to the architecture or the input modality.

Table 4.5 presents the validation and test accuracies for I3D and F3D on all sign datasets. In our training experiments, I3D consistently outperformed F3D in all metrics and datasets. This drop in accuracy for F3D can be attributed to several factors, including the considerable amount of noise in the extracted optical flow and the loss of detailed spatial information inherent in optical flow, such as the configuration of individual fingers, which often appear as a single undifferentiated blob in the flow representation. For WLASL 100 and WLASL 2000, the drop in accuracy was greater ($5-6$ percentage points), likely due to the lower image quality of the videos in WLASL, which in turn degrades the quality of the extracted optical flow. Unlike AUTSL and MSASL, where videos are resized during training when the batch is formed, videos in WLASL have been originally lossy compressed and saved as MP4 after being resized down to 256x256, causing the lower image quality.

Table 4.5: Comparison of I3D and F3D validation and test accuracies on different sign datasets. Δ is the accuracy difference between I3D and F3D.

Dataset	Validation Accuracy			Test Accuracy		
	I3D	F3D	Δ	I3D	F3D	Δ
AUTSL	**94.14**	92.01	2.13	**94.04**	91.48	2.56
MSASL-100	**80.67**	78.69	1.98	**83.38**	82.58	0.80
MSASL-1000	**57.88**	55.13	2.75	**61.26**	59.09	2.17
WLASL-100	**78.11**	77.22	0.89	**77.52**	68.99	8.53
WLASL-2000	**48.31**	43.13	5.18	**46.77**	39.89	6.88

However, the underperformance on all datasets should not, in itself, disqualify F3D from being potentially useful as a complimentary stream to the RGB stream, as there are instances where F3D correctly classifies samples that I3D does not. What we aim to demonstrate is that the reason F3D classifies these samples correctly, while I3D does not, is not because motion is particularly discriminative for those classes.

Figure 4.4 illustrates this by comparing the difference in accuracy per class between I3D and F3D on the validation set of AUTSL. Each bar represents (I3D accuracy) – (F3D accuracy). Positive bars indicate the classes where I3D outperformed F3D, while negative bars show the opposite. Absence of bar indicates that both models classified an equal portion of samples in the class correctly.

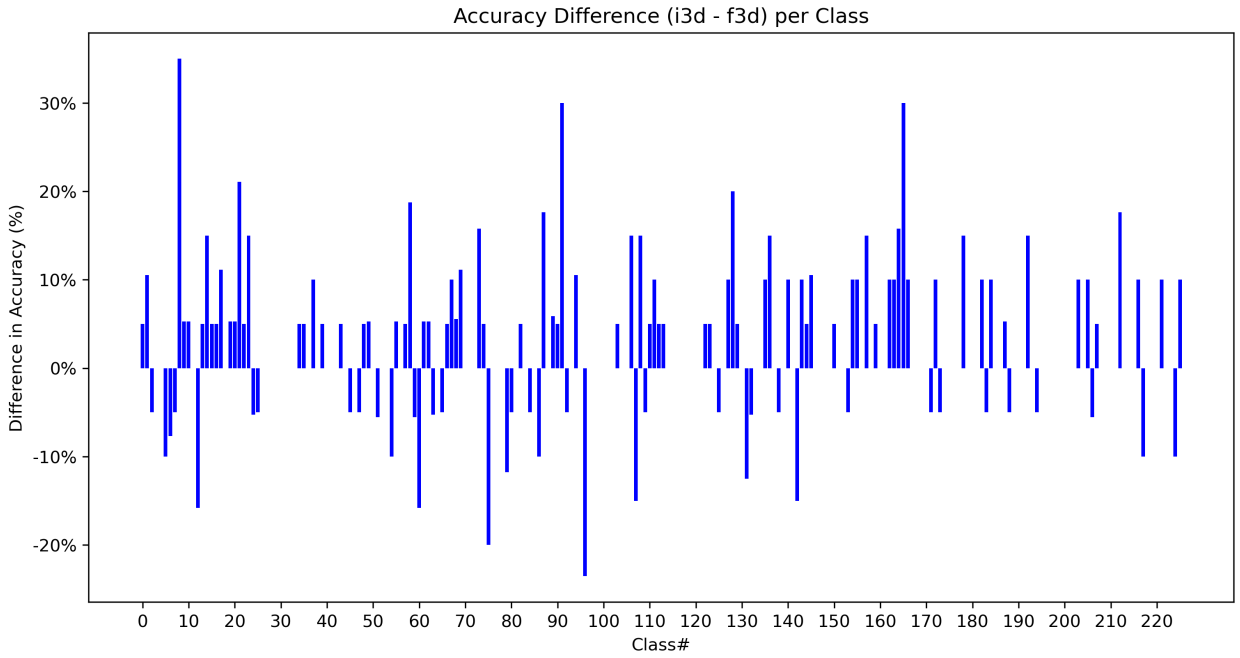


Figure 4.4: Accuracy difference (I3D - F3D) per class on AUTSL validation set

As shown by the length of the negative bars in Figure 4.4, F3D outperformed I3D by a margin of 20% to 23.6% in only two classes: #96 “medicine” and #75 “shirt”. However, using the binomial exact test, this difference is not statistically significant, with p-values of 0.21875 and 0.125 for classes #96 and #75, respectively, exceeding the significant level

of 0.05. More importantly, the instances misclassified by I3D in these two classes were not due to motion patterns being a key discriminative factor, as seen in classes #16 and #47 discussed in Section 4.2.

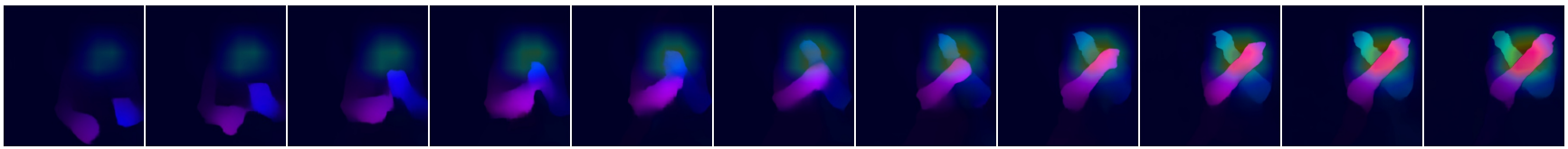
A key reason some instances are classified correctly by F3D but incorrectly by I3D is the disregard of irrelevant features by optical flow, such as background or clothing. This can be clearly demonstrated by their performance on class #107 "crossroads", where F3D classified all instances correctly while I3D misclassified three instances (resulting in a 15% difference in accuracy). These three instances were all confused with class #123 "hug". Both "crossroads" and "hug" have similar motion patterns, but they differ in the shape of the hands, where the hands are open for "hug" and closed for "crossroads".

These three instances were all performed by the same signer wearing a black faux fur vest. To visualize the region of the input influencing I3D's incorrect classification of these instances, we applied the Grad-CAM technique to both I3D and F3D for comparison (Figure 4.5). As shown in the figure, most of the activation for class #107 in I3D corresponds to the black vest region. Since this region is ignored by optical flow in F3D due to very low motion, class #107 was correctly activated by the region between the hands in F3D. To further investigate whether this influence occurred by chance, we trained I3D on AUTSL with masked input, allowing only the region around the two hands in all input frames to appear. The bounding boxes of the two hands were extracted by a pretrained HRNet [33] for its better skeleton detection accuracy compared to Mediapipe. This masked-RGB-input I3D achieved 92.94% accuracy on the validation set, a drop of 1.2 percentage points from the full-RGB-input I3D. However, the new masked-I3D correctly classified all instances of class #107, including those misclassified by I3D. As Grad-CAM shows in Figure 4.5, by masking the irrelevant feature in the input (the black fur vest in this case), the class activation is triggered by the expected region. This example supports our hypothesis that the filtering of irrelevant features by optical flow explains why it outperforms I3D with full RGB input in some instances.

RGB frames of a "crossroads" sign, incorrectly classified by I3D

Grad-CAM shows that the misclassification was influenced by the black fur vest

Grad-CAM shows that I3D correctly focuses on the region of interest (hand crossing) when masking is applied



Grad-CAM shows that F3D correctly focuses on the region of interest (hand crossing) when using optical flow

Figure 4.5: A sample video from class #107 "crossroads", which is classified correctly by F3D but incorrectly by I3D. Grad-CAM visualization on I3D (2nd row) reveals that the misclassification was influenced by the black fur vest. Masking irrelevant regions explicitly in the RGB input (3rd row) or implicitly using optical flow (4th row) forces the network to attend to the correct region in the input (the crossing of the hands).

To summarize this section (4.3.1), while F3D underperformed I3D in overall accuracy, its better performance in certain classes is not statistically significant. Even if we assume it was significant, these classes are not characterized by motion patterns being a key discriminative factor. Therefore, explicit motion learning cannot be considered as the reason for this improvement. Instead, the role of optical flow can be viewed as filtering out irrelevant features, much like the process of masking the input video.

4.3.2 Effect of Motion Components in Optical Flow

Given that training on the optical flow representation of sign videos did not result in improvements for specific classes where motion patterns are expected to be important, this raises two key questions.

1. Is retaining all motion components in optical flow necessary to maintain the same level of overall accuracy?

2. What aspect of optical flow representation contribute meaningfully to the overall accuracy of sign recognition?

To answer these questions, we conduct ablation study on AUTSL dataset by training F3D on different components of the optical flow, including magnitude only, direction only, and masked (thresholded) magnitude.

The optical flow extracted by the RAFT model [56] consists of two channels, which represent the horizontal (x) and the vertical (y) components of the estimated movement from $frame_t$ to $frame_{t+1}$ at each pixel location. An alternative but equivalent representation is the magnitude-direction format, often used for visualizing optical flow, where magnitude is represented by pixel intensity and the direction (angle) by color. The magnitude and direction can be calculated from the X-Y format as follows:

$$\text{Magnitude} = \sqrt{(v_x^2 + v_y^2)} \tag{4.5}$$

$$\text{Direction} = \theta = \arctan\left(\frac{v_y}{v_x}\right) \tag{4.6}$$

where v_x and v_y represent the horizontal and vertical components of the flow, respectively, and θ is the direction (angle) of the movement in radians.

In this ablation study, we compare the classification accuracy of five models.

1. F3D with both input channels for horizontal and vertical components.
2. F3D with both input channels for magnitude and direction of motion.
3. F3D with single input channel for magnitude of motion.
4. F3D with single input channel for masked (binary) magnitude.
5. F3D with single input channel for direction of motion

Figure 4.6 shows examples of input representation for each of these models. The training of these five models differs from the F3D model in Section 4.3.1 in the following aspects.

- For a fair comparison between the five models, they are trained from scratch, without pretraining on Kinetics dataset, due to lack of availability of pretrained single-channel F3D model.
- Since training F3D models is very time-consuming and resource-intensive due to the computational cost of the large RAFT, we train on 224x224 crops sampled normally from 384x384 resized frames from the original resolution, instead of training on 256x256 crops sampled normally from 512x512. During inference, we measure accuracy using 256x256 center crop instead of 320x320 center crop.

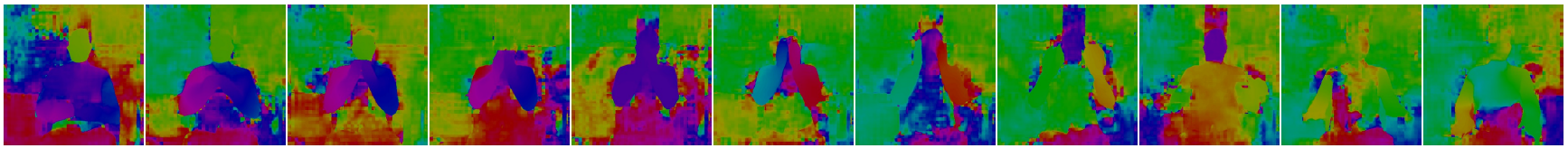
All other aspects of the training pipeline remain the same, including the number of frames in a crop (64), the batch size (32), the number of epochs (40), the learning rate (1×10^{-3}), and Adam optimization with one-cycle scheduling. Table 4.6 shows the results of the models on the test set of AUTSL.

RGB frames of a "garden" sign

Optical flow showing both magnitude (visualized as intensity) and direction (visualized as color)

Optical flow magnitude only

Optical flow magnitude converted to a binary mask (magnitude threshold = 0.5)



Optical flow direction only

Figure 4.6: Optical flow visualizations of a sample video from class #21 "garden". Motion direction (angles) is visualized as color while motion magnitude as pixel intensity. The primary value of optical flow in SLR is its masking (segmentation) property.

Table 4.6: Performance comparison on AUTSL test set of F3D models trained on different flow input channels.

Input Flow Channels	Per-instance	Per-class	Top-5	p-value
X and Y (baseline)	**85.52**	**85.43**	**98.05**	-
Magnitude and Direction	**85.14**	**85.02**	**98.08**	0.4389
Magnitude only	**85.7**	**85.58**	**98.02**	0.7424
Masked Magnitude only	82.58	82.48	96.39	5.24×10^{-7}
Direction only	81.59	81.53	96.71	4.03×10^{-12}

The results in Tables 4.6 shows that removing the direction channel and using only the magnitude (speed) channel does not change the overall accuracy compared to the baseline, and any small difference is likely due to chance (p-value=0.7424). This is a significant finding, yet it is counter intuitive for two reasons. First, direction information conveys the trajectory of hand movement, which is crucial for distinguishing between signs that may have similar shapes or positions but different movement paths. Second, the speed of motion (magnitude of displacement per two consecutive frames) is rarely as distinguishing factor as direction of motion. Despite this, the experiment show that classification based solely on magnitude (speed) yields the same overall accuracy (about 85%) as the baseline, while classification based solely on direction noticeably degrades accuracy, achieving only 81.59%. This casts doubt on the notion that optical flow is useful for sign recognition due to its explicit representation of motion since the removal of the most important component of motion (direction) did not impact the accuracy.

Our hypothesis to explain why classification with only magnitude performs as well as classification using both X and Y channels (baseline) or both magnitude and direction is that magnitude effectively indicates where the motion occurs in the frame. Such information is what is most relevant to sign recognition because static objects, such as the background or non-moving parts of the body, are often not relevant to sign identification. To test this hypothesis, we convert the magnitude channel into a binary mask, disregarding all speed information from the pixels. In this mask representation, a value of 1 indicates that motion

occurs at a pixel, while a value of 0 indicates the absence of motion (Figure 4.6). During training, a threshold is chosen randomly for each video at every training step from the interval $[0, 1]$ and applied to the magnitude channel to obtain the binary mask channel, such that:

$$\text{mask}(p) = \begin{cases} 1, & \text{if magnitude}(p) > \text{threshold} \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

where p represents the pixel. During inference (test), we use a constant threshold value of 0.5 for all videos.

As the result in Table 4.6 shows, training F3D on the binary mask channel alone achieves an overall accuracy of 82.58%, not too far from the full optical flow baseline (85.52%). This provides strong evidence that the primary value of optical flow in sign recognition lies in its masking properties. In other words, its ability to filter out irrelevant pixels, which in sign language typically corresponds to those with little or no motion.

This masking hypothesis can also explain why training on the flow direction channel alone achieves 81.59%, when it was expected to perform much worse, given that magnitude alone achieves similar performance to magnitude + direction, suggesting the direction channel adds no value to the classification. However, from the perspective of the masking hypothesis, the direction channel can be viewed as a way to "segment" a moving region (such as a moving arm) since pixels in that region move in the same direction and thus have similar angle values (visualized as similar color in last row of Figure 4.6). This consistency in direction provides a strong signal to the convolutional network to distinguish the moving region from the inconsistent noisy background (with varying angle values visualized as different colors).

In conclusion, the ablation study in this section further supports the findings from Section 4.3.1, providing additional evidence for our proposed masking hypothesis: that the primary value of optical flow lies in its ability to segment relevant regions for sign recognition (i.e. moving regions) rather than improved learning of motion patterns. The 3D convolutional architecture used in models like F3D, I3D, or R3D already effectively learns motion

patterns from the sequence of input frames, regardless of input modality—whether RGB, depth, or optical flow. This finding is significant because achieving similar segmentation functionality through computationally cheaper methods than optical flow could improve the efficiency of multi-stream architecture for sign recognition without compromising accuracy.

4.4 Results of Skeleton-Based Sign Recognition

In Sections 4.2 and 4.3, we analyzed the impact of 3D convolutional networks (slow fusion) on sign recognition using RGB input and optical flow extracted from RGB, respectively. With raw RGB input, no a priori assumption is made about which pixels in the input are most relevant to sign recognition. However, with optical flow input, the neural network is trained under the assumption that the most relevant pixels for sign recognition are those that are in motion, regardless of any other properties (e.g., color).

Skeleton-based sign recognition, on the other hand, operates under the assumption that the most relevant features for sign recognition are the relative positions of body joints and how their positions change over time. To contrast skeleton data with optical flow, skeleton can be viewed as an extreme form of segmentation, where the key features are individual points (e.g., joint locations) rather than broader regions of pixels (e.g., the arm region). Unlike optical flow, skeleton-based recognition often benefits from a much lower computational cost in the second stage since it processes a defined set of joints rather than dense pixel data.

4.4.1 Performance of Skeleton Models

In this section, we examine the performance of structured and unstructured representations of skeleton points within each frame. For the unstructured representation, we use a recurrent neural network (GRU) and a Transformer Encoder, where the coordinates of all 31 joints in a frame are flattened into a single feature vector. As a result, each frame is represented as a bag of features, with no prior knowledge provided to the model about the relationships or connectivity between the joints. We consider the Gated Recurrent Unit

(GRU) a solid baseline for comparison with other advanced skeleton-based models due to its simplicity. We also experimented with bi-directional GRU but did not observe improvement in accuracy over the standard GRU.

In the structured representation, each coordinate is associated with its respective joint, and each joint is linked with its neighboring joints based on the skeleton connectivity of the human body. Graph Convolutional Networks (GCN) are well-suited for this representation, where joints are represented as nodes and their connections as edges. Specifically, we use Spatio-Temporal Graph Convolutional Network (ST-GCN) [85], developed originally for action recognition, to learn both the temporal features across frames (similar to RNN and Transformers) and the spatial features within each frame.

Tables 4.7, 4.8, and 4.9 show the top-1 accuracy of skeleton-based models on the validation and test sets of AUTSL, MSASL, and WLASL, respectively, with no pretraining. The Spatio-Temporal Graph Convolutional Network (ST-GCN) consistently outperforms the GRU baseline by a significant margin in all datasets with the exception of WL100, where its improvement is limited potentially due to the smaller number of training samples in this split compared to others.

In contrast, the transformer encoder with unstructured representation outperformed the GRU baseline only on the AUTSL dataset while achieving the worst performance on all other datasets. We experimented with various hyperparameters, including the number of layers, number of heads, feature dimensionality, dropout probability, and weight decay regularization, but these adjustments did not lead to further improvement in the transformer model's accuracy.

Transformer encoders rely on a self-attention mechanism to model all pairwise interactions in a sequence of frames. While this mechanism makes transformers more flexible, it also makes them more "data-hungry" and, hence, more susceptible to overfitting when the number of training samples is small. This explains why the transformer encoder outperformed GRU only in AUTSL, which has an abundant amount of training samples per class.

Table 4.7: Results of skeleton models on AUTSL dataset. I.O.B. denotes improvement over baseline (GRU).

Model	Validation		Test	
	Top-1	I.O.B.	Top-1	I.O.B.
GRU (baseline)	85.08	0	81.53	0
Transformer Enc.	89.18	+4.10	85.25	+3.72
ST-GCN	**92.91**	+7.83	**92.49**	+10.96

Table 4.8: Results of skeleton models on MSASL-100 and MSASL-1000 datasets

Model	MSASL-100				MSASL-1000			
	Validation		Test		Validation		Test	
	Top-1	I.O.B.	Top-1	I.O.B.	Top-1	I.O.B.	Top-1	I.O.B.
GRU (baseline)	69.97	0	76.99	0	45.57	0	48.96	0
Transformer Enc.	68.38	-1.59	71.14	-5.85	44.95	-0.62	48.53	-0.43
ST-GCN	**76.81**	+6.84	**79.92**	+2.93	**54.15**	+8.58	**58.22**	+9.26

Table 4.9: Results of skeleton models on WLASL-100 and WLASL-2000 datasets

Model	WLASL-100				WLASL-2000			
	Validation		Test		Validation		Test	
	Top-1	I.O.B.	Top-1	I.O.B.	Top-1	I.O.B.	Top-1	I.O.B.
GRU (baseline)	76.63	0	71.32	0	41.24	0	39.33	0
Transformer Enc.	74.56	-2.07	69.77	-1.55	38.51	-2.73	36.59	-2.74
ST-GCN	**78.11**	+1.48	**73.64**	+2.32	**49.03**	+7.79	**48.37**	+9.04

In contrast, GRU processes data in a recurrent fashion, updating each hidden state from the previous hidden state and sharing the model's parameters across all time steps. This inductive bias constrains the GRU model's flexibility but enables more efficient learning from smaller datasets.

ST-GCN uses an adjacency matrix to represent the connectivity of joints in the human skeleton, hence introducing an inductive bias by limiting connections to physically relevant joints (sparse representation). Moreover, sharing weights in spatial convolutions across nodes and temporal convolutions across frames further enhances its inductive bias. The fact that

ST-GCN architecture outperforms transformers and GRU in both cases of many training samples per class (as with AUTSL dataset) and a relatively few training samples per class (as with MSASL and WLASL datasets) strongly suggests the usefulness of the structured representation and the inductive bias in ST-GCN for skeleton-based sign recognition.

4.4.2 Impact of Temporal Aspects in ST-GCN

In Section 4.2 we demonstrated the significant impact of slow fusion (e.g., 3D convolution) over late fusion (2D convolution + average pooling at the end) for RGB-based sign recognition. We can perform a similar type of ablation study on ST-GCN by reducing the kernel size of its temporal convolution from 9 to 1, effectively eliminating convolution across time. This spatial-only GCN processes each frame's skeleton separately, then applies average pooling across all nodes and all frames before the classification layer. By removing the temporal convolution, the model's accuracy drops significantly from 92.91% to 81.21%. This demonstrates that slow fusion is just as crucial for skeleton-based sign recognition as it is for RGB-based sign recognition. Fortunately, retaining the temporal convolution in ST-GCN is much less computationally costly than keeping the temporal convolution in 3D ResNet.

In addition to training on skeleton data, ST-GCN can also be trained on skeleton motion data with no change to its architecture. Skeleton motion data represents the amount of displacement of each joint between every two consecutive frames, similar to how optical flow represents the amount of displacement of each pixel between every two consecutive frames. Skeleton motion data is calculated from skeleton data by subtracting joints' coordinates of frame t from frame $t + 1$, which is a much simpler and straightforward process than computing optical flow from RGB frames. This makes skeleton-motion-based recognition an attractive choice as a second stream to skeleton-based models for action recognition as proposed in [102]. In our experiment, training ST-GCN model on skeleton motion achieves 90.92% accuracy on AUTSL dataset, a drop of only 2 percentage points from the ST-GCN model in Table 4.7. However, this drop in accuracy becomes significantly larger in MSASL

and WLASL, achieving worse accuracy than the transformer encoder. Therefore, we do not use the skeleton motion stream due to its inconsistent performance across the datasets.

4.4.3 Comparison with RGB models

One potential cause for this inconsistency is the quality of the skeleton extracted by the first stage (MediaPipe module), where hand detection is sometimes missed (false negative). This occasionally happens in AUTSL dataset (see the last example in Figure 3.8, Chapter 3), but it occurs more frequently in MSASL and WLASL, as they have lower data quality than AUTSL. In general, skeleton models tend to underperform RGB models in sign recognition for two reasons.

1. Missed hand detection and imprecise localization of skeleton joints, especially when using lightweight, real-time modules such as MediaPipe. Even if, in theory, the ST-GCN model achieves optimal learning (minimal bias and variance errors), the measurement noise in the extracted skeleton data likely contributes meaningfully to the model's error.

2. RGB models (e.g., R3D and I3D) are less affected by input distortions than skeleton models, as they rely on rich visual cues from entire regions of pixels (e.g., hand) rather than individual points. However, RGB models face a different kind of noise in the form of irrelevant input features, such as background objects, clothing, or hair color. As a result, RGB models must do the additional work of filtering out these distractions to extract the relevant signal. In contrast, ST-GCN receives a more direct estimation of the signal relevant to the task (the skeleton), but this makes it highly dependent on the quality of the input skeleton.

Therefore, combining the predictions from both RGB and skeleton models can mitigate some of the weaknesses in each stream. We show the results and analysis of this combination later in Section 4.6.

4.4.4 Interpretability through Edge Importance Weighting

The adjacency matrix of ST-GCN, which describe how the nodes in the graph are connected, is hard-coded based on the 31 skeleton landmarks we selected as the most relevant for sign recognition. However, not all edges in the adjacency matrix are equally important for sign recognition. Therefore, we apply edge importance weighting to allow the model to learn the appropriate weighting of the edges between the nodes in the graph. While this approach has been utilized to improve the performance of ST-GCN [85], we are not aware of it being used as a visualization tool in the sign language and action recognition literature. To address this gap, we propose the following method for visualizing the edge importance weighting to enhance the interpretability of ST-GCN.

Let the normalized adjacency matrix $A_k \in \mathbb{R}^{N \times N}$ represent the spatial connections between N joints in the skeleton (there are 31 joints) for each spatial partition k (e.g., root, close, and further connections). For each layer l of the ST-GCN model (there are 10 layers), we also have learnable edge importance weighting matrices $E_k^{(l)} \in \mathbb{R}^{N \times N}$, where k refers to the spatial partition and l refers to the layer.

The weighted adjacency matrix $\hat{A}_k^{(l)}$ is computed by element-wise multiplication of the normalized adjacency matrix A_k by the edge importance weights $E_k^{(l)}$ for that layer and partition. This can be expressed as:

$$\hat{A}_k^{(l)} = A_k \odot E_k^{(l)} \tag{4.8}$$

where $\odot$ denotes the element-wise product.

To calculate the final edge importance for visualization purposes, the weighted adjacency matrices are summed across all 10 layers and all 3 spatial partitions. This is expressed in one step as:

$$I_{\text{final}} = \sum_{k=1}^{3} \sum_{l=1}^{10} \left(A_k \odot E_k^{(l)} \right) \tag{4.9}$$

We use the matrix $I_{\text{final}} \in \mathbb{R}^{N \times N}$ for visualization, with each entry $I_{\text{final}}[j, m]$ representing the total importance of the connection between joint j and joint m.

Figure 4.7 visualizes this matrix I_{final} as heat values between joints overlaid on a sample frame from class #99 "push" in the AUTSL dataset. This weighting, however, is not specific to a particular sign but represents a summary of what the model has learned as important for the classification of signs across the entire training set.

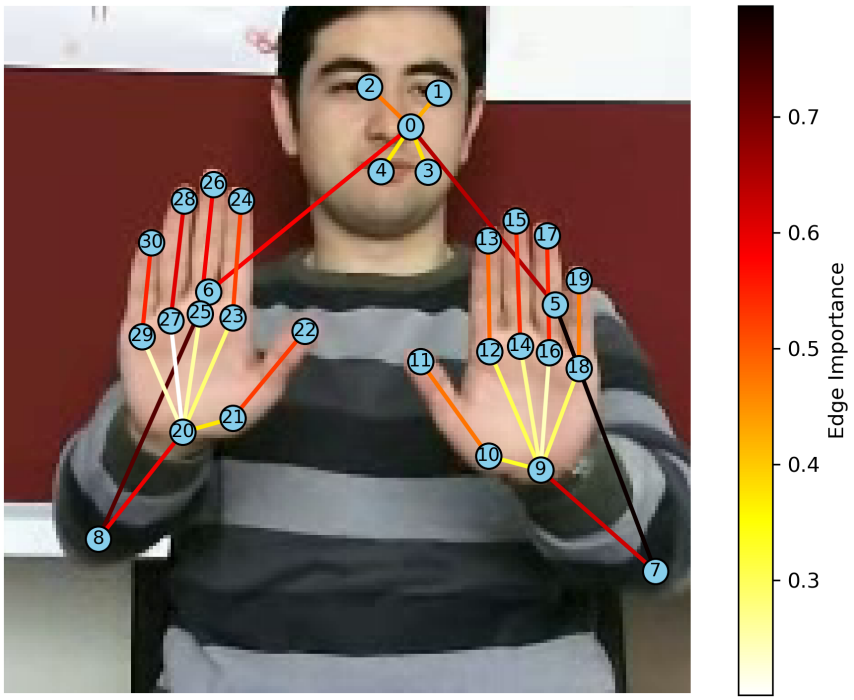


Figure 4.7: Visualization of the final edge importance matrix summarizing edge importance in the dataset (mirrored image), where darker edges indicate higher importance.

Using the visualization of edge importance in Figure 4.7, we learn the following characteristics of sign recognition in AUTSL dataset.

1. The weights from the base of fingers to the tip of fingers (i.e. right-hand edges: [10,11], [12,13], [14, 15], [16,17], [18,19] and left-hand edges: [21,22], [23,24], [25,26], [27,28], [29,30]) are more important for sign recognition than the weights from the wrist to the base of fingers (i.e. node 9 to nodes 10, 12, 14, 16, 18 and node 20 to nodes 21, 23,

25, 27, 29). This confirms that the learned graph representation in the ST-GCN model matches the expected reality as the configuration of fingers is much more discriminative for sign recognition than bones of the palm, whose movements are highly correlated together.

2. The weights from the shoulders to elbows (edges [5, 7] and [6, 8]) are larger than the weights from the elbows to wrists (edges [7, 9] and [8, 20]). This suggests that arm movement is more significant for sign recognition than forearm movement. Moreover, the model has learned to weigh the right arm (edge [5, 7]) more than the left arm (edge [6, 8]), which is likely due to most signers being right-handed and performing one-handed signs with their right hand while the left hand is at rest.

3. Edges [0, 1] and [0, 2] (nose to eyes) are weighted more than edges [0, 3] and [0, 4] (nose to mouth). This finding is unexpected because the distance between the nose and eyes does not change across the video frames, as it is a fixed property of the face. However, one possible explanation is that the model might have learned to use edge [0, 2] to infer other useful properties about the face positioning, such as whether the face is tilted as performed in the sign for "pillow" (class #214).

While this visualization method provides valuable insights into the overall weighting of edge importance across the entire sign language dataset, therefore improving model interpretability, further insights could be gained from identifying the specific joints or bones that contribute most significantly to the activation of individual classes, akin to the CAM visualization for RGB and optical flow models. This approach could be useful for debugging to better understand why the ST-GCN model misclassifies certain skeleton inputs. We leave this for future work.

4.5 Improving Performance with Transfer Learning

A key aspect of the training methodology we propose in this dissertation is hierarchal transfer learning to improve the model's performance. In our experiments, we consistently observe better accuracy on the target sign dataset when initializing the model's weights using a pre-trained model from Kinetics—a large action recognition dataset— than by random weight initialization. This finding aligns with established results in the action recognition literature. However, we also introduce an intermediate level of pre-training, where the model is subsequently trained on a large-scale sign recognition dataset (distinct from the target dataset) before being trained on the target sign dataset. Our approach is inspired by Carreira et al. [2], where the model was pretrained on both ImageNet and Kinetics datasets before the target action dataset. We refer to this three-step training process as coarse-to-fine training. In summary, the model is

1. initially trained on a general action recognition dataset (e.g., Kinetics),
2. then trained on a large sign language dataset (distinct from the target dataset),
3. and finally, trained on the target sign language dataset intended for model evaluation.

The rationale for this progression is to first allow the model to learn fundamental spatial and temporal features useful for general action recognition tasks. Then, the model adapts these spatiotemporal features to learn higher-level, domain-specific features for gesture recognition. Lastly, the model is fine-tuned to learn the specific nuances of the target sign language.

4.5.1 Ablation Study

The ablation study we provide in Tables 4.10 and 4.11 examines the effect of different components of this multi-level training process. Table 4.10 shows that training I3D on Kinetics400 or AUTSL improves the accuracy by a significant margin (about 12 percentage points) compared to training with random weight initialization (first row). However, pretraining on

Kinetics, followed by AUTSL, results in the best accuracy on MSASL1000, achieving 66.90% on its test set.

Table 4.10: Ablation of transfer learning from Kinetics and/or AUTSL to MSASL-1000

Trained on Kinetics400	Trained on AUTSL	Trained on MSASL1000		
		Per-instance accuracy	Per-class accuracy	Top-5 accuracy
✗	✗	48.24	43.87	70.27
✓	✗	61.26	57.29	80.74
✗	✓	61.60	58.77	81.46
✓	✓	**66.90**	**63.45**	**84.69**

Table 4.11: Ablation of transfer learning from Kinetics and/or MSASL-1000 to AUTSL

Trained on Kinetics400	Trained on MSASL1000	Trained on AUTSL		
		Per-instance accuracy	Per-class accuracy	Top-5 accuracy
✗	✗	91.56	91.56	98.96
✓	✗	94.04	94.07	99.36
✗	✓	89.82	89.79	98.08
✓	✓	94.52	94.53	99.25

In Table 4.11, we swap the intermediate and target sign language datasets. Pretraining I3D on Kinetics still provides a noticeable improvement for AUTSL (about 2.5 percentage points) over no pretraining. However, pretraining on MSASL1000, either alone or after Kinetics, does not yield statistically significant improvements in AUTSL accuracy. This is likely due to the low density (number of training videos per class) in MSASL1000. Although MSASL1000 has 1000 classes, which is significantly more than Kinetics (400 classes) and AUTSL (226 classes), MSASL1000 has an average of 14 training videos per class—much fewer than AUTSL (124 training videos per class) and Kentics400 (600 training videos per

class). The low training samples per class in MSASL1000 caused I3D to overfit its training set, reducing the usefulness of the learned features for generalization and transfer learning. Therefore, for coarse-to-fine transfer learning to show meaningful improvement, we recommend pretraining on datasets—in the first and second steps—that have both a large total number of training samples (high mass) and a high number of samples per class (high density).

To further analyze the impact of pretraining on test accuracy as a function of the number of training samples per class, we grouped the 1000 classes in MS-ASL datasets into five bins based on their number of training samples (Figure 4.8). Each bin contains three bar charts representing different pretraining conditions: no pretraining, pretraining on Kinetics, and pretraining on Kinetics followed by AUTSL. These bar charts show the average accuracy for all classes whose number of training samples falls within the corresponding range: [0, 10), [10, 20), . . . , [40, 50).

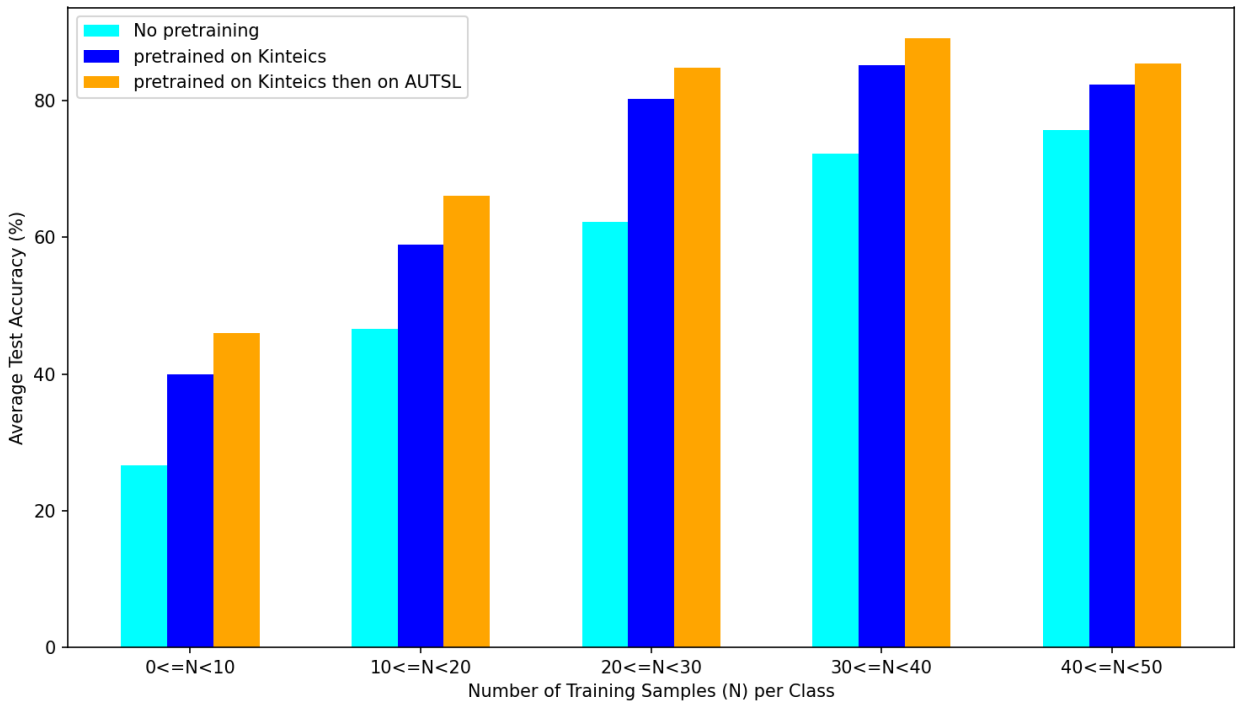


Figure 4.8: Average test accuracy vs number of training samples per class (MS-ASL1000 dataset)

The following key observations can be drawn from Figure 4.8.

1. The test accuracy of a class, on average, is positively correlated with the number of training samples in the class, regardless of the pretraining condition. This explains why the overall test accuracy of MSASL1000 is moderate, as the majority of classes in the dataset (78%) fall within the first two bins: [0, 10) and [10, 20).

2. Pretraining, whether on Kinetics or Kinetics+AUTSL, constantly improves the average accuracy across all bins compared to no pretraining. However, the improvement in average accuracy for the last bin [40, 50) is less significant than for earlier bins. This suggests a diminishing return from transfer learning as the number of training samples per class increases.

3. Similarly, pretraining Kinetics+AUTSL consistently improves the average accuracy over pretraining on Kinetics alone. This aligns with the findings from the ablation study in Table 4.10. A similar diminishing return can be observed between the blue and orange bars, where the earliest bins benefit more, on average, from additional pretraining on AUTSL than later bins.

In summary, the results shown in Figure 4.8 illustrate the important role of the number of training samples per class on test accuracy and on the margin of improvements introduced by transfer learning.

4.5.2 Results of All Models

The results discussed in Section 4.5.1 pertain to a single model (I3D) using one modality (RGB). Table 4.12 summarizes the accuracy results on AUTSL dataset for all models. In this table, I3D, R3D, and F3D are pretrained on Kinetics only, while ST-GCN is trained from random weight initialization due to the lack of available Kinetics weights for ST-GCN. To evaluate whether the benefit of pretraining on AUTSL dataset, in addition to Kinetics, extends to other modalities such as optical flow and skeleton, we compare the test accuracy

Table 4.12: Test accuracy of all models on AUTSL dataset

Model	Per-instance	Per-class	Top-5
RGB → I3D	94.04	94.07	99.36
RGB → R3D	95.38	95.38	99.36
RGB → Flow → F3D	91.48	91.38	98.80
RGB → Skeleton → ST-GCN	92.36	92.26	98.77

Table 4.13: Test accuracy of all models on the MSASL-100 and MSASL-1000 datasets. [✗], [✓], and Δ denote results without pretraining on AUTSL, with pretraining on AUTSL, and the accuracy difference between them, respectively.

Model	MSASL-100					MSASL-1000				
	Per-instance		Per-class			Per-instance		Per-class		
	✗	✓	✗	✓	Δ	✗	✓	✗	✓	Δ
I3D	83.38	85.77	83.69	86.55	+2.86	61.26	66.90	57.29	63.45	+6.16
R3D	83.78	86.17	84.30	87.26	+2.96	61.04	65.72	56.93	62.58	+5.65
F3D	82.58	86.97	82.94	87.43	+4.49	59.09	61.07	55.36	57.30	+1.94
ST-GCN	80.85	83.24	82.11	84.17	+2.06	60.58	63.93	57.73	62.53	+4.80
Avg Δ					+3.09					+4.64

Table 4.14: Test accuracy of all models on WLASL-100 and WLASL-2000 datasets. [✗], [✓], and Δ denote results without pretraining on AUTSL, with pretraining on AUTSL, and the accuracy difference between them, respectively.

Model	WLASL-100					WLASL-2000				
	Per-instance		Per-class			Per-instance		Per-class		
	✗	✓	✗	✓	Δ	✗	✓	✗	✓	Δ
I3D	77.52	82.56	78.02	82.55	+4.53	46.77	52.88	43.80	50.32	+6.52
R3D	79.07	80.23	79.80	80.17	+0.37	47.78	53.09	45.24	51.08	+5.84
F3D	68.99	75.97	68.97	76.30	+7.33	39.89	42.98	37.09	40.27	+3.18
ST-GCN	77.13	78.68	78.00	79.38	+1.38	49.10	52.57	46.49	50.18	+3.69
Avg Δ					+3.40					+4.81

of each model with (✓) and without (✗) pretraining on AUTSL, as shown in Tables 4.13 and 4.14.

Tables 4.13 and 4.14 show that the additional training step on AUTSL improves accuracy for both RGB models (I3D and R3D), the optical flow model (F3D), and the skeleton model (ST-GCN) on both the MSASL and WLASL datasets, as well as their respective splits. The effect of diminishing returns is noticeable, as the higher-density splits (MSASL-100 and WL-ASL100) show average improvements of $2-3$ percentage points, which is slightly lower than the 4 percentage points average improvement observed for the lower-density splits (MSASL-1000 and WLASL-2000).

The results in this section demonstrate that feature learning can effectively transfer across different sign languages. In this case, the features learned from AUTSL (a Turkish sign language dataset) were beneficial for learning features in American sign language datasets such as MSASL and WLASL, across various modalities. We intentionally avoided transfer learning between the two American sign language datasets (MSASL and WLASL) because some videos in the training set of one dataset appear in the test set of the other. This overlap could falsely suggest improved accuracy due to transfer learning, when in reality, the gains would result from training and testing on the same instances.

The four models shown in Table 4.12, as well as those with additional pretraining on AUTSL (✓) in Tables 4.13 and 4.14, will be used to construct and evaluate multi-stream networks, as discussed in the following sections.

4.6 Results of Two-Stream Networks

By comparing the results of single-stream 3D convolutional RGB models (I3D and R3D), the optical flow model (F3D), and the graph convolutional skeleton model (ST-GCN), we observe that pre-trained RGB models almost consistently outperform optical flow and skeleton models across different datasets and splits. This raises the question of whether using one or two RGB models is sufficient for sign recognition, or if there is a unique value in combining

multiple modalities that justifies the extra computational overhead—a key research question of this dissertation. In particular, this section investigates the following questions.

1. How much improvement in accuracy over the single-stream baseline is achieved by adding additional streams to the network? Does this gain justify the added computational cost?

2. What combination of streams consistently yields the best accuracy for sign recognition, and how does that relate to the error correlation between the streams?

3. Do the results support the hypothesis that certain signs are better recognized by a specific stream/modality?

To address these questions, we begin by examining two-stream networks, which integrate predictions from two models. This analysis will then extend to three-stream networks in Section 4.7, providing a comprehensive view of all possible combinations.

The stream fusion results presented previously in Chapter 3 (Table 3.2) are based on a specific stream combination (GCN and I3D models) for a single dataset (MSASL-1000). However, there are ${}^4C_2 = 6$ possible combinations to form a two-stream network from the four models: I3D, R3D, F3D, and ST-GCN. We evaluate all two-stream combinations across all datasets to gain better confidence in any conclusions.

Tables 4.15, 4.16, and 4.17 report the test results of the six possible two-stream networks formed from the four models on AUTSL, MSASL, and WLASL datasets, respectively. All listed α^* values were found using grid search to maximize the validation accuracy. We do not use the training set to search for the fusion weights α^* because the training accuracies of all models are already high. We use the I3D model as our single-stream baseline due to its high accuracy and relatively lower computational cost compared to other models. For any two-stream network, its Improvement Over Baseline (I.O.B.) metric is thus defined as the difference between its accuracy and the I3D model accuracy. The reduction in error rate

Table 4.15: Test accuracy of two-stream networks on AUTSL dataset. I.O.B. denotes the improvement over the single-stream baseline (I3D). % Error represents the reduction in the error rate compared to the baseline (I3D). α^* is the fusion weight for the first stream. "Corr." refers to the error correlation between the two streams.

Streams		AUTSL					
#1	#2	α^*	Top-1	I.O.B.	% Error ↓	Top-5	Corr.
I3D	-	-	94.04	-	-	99.36	-
I3D	R3D	0.45	95.88	+1.84	30.94	99.76	0.58
F3D	I3D	0.45	94.87	+0.83	13.90	99.57	0.50
F3D	R3D	0.36	96.26	+2.22	37.22	99.68	0.42
GCN	I3D	0.27	96.50	+2.46	41.26	99.71	0.31
GCN	R3D	0.17	97.19	+3.15	52.91	99.73	0.29
GCN	F3D	0.45	95.91	+1.87	31.39	99.76	0.32

Table 4.16: Test accuracy of two-stream networks on MSASL-100 and MSASL-1000 datasets

Streams		MSASL-100					MSASL-1000				
#1	#2	α^*	Top-1	I.O.B.	Top-5	Corr.	α^*	Top-1	I.O.B.	Top-5	Corr.
I3D	-	-	85.77	-	95.88	-	-	66.90	-	84.69	-
I3D	R3D	0.46	88.56	+2.79	97.34	0.63	0.60	70.23	+3.33	87.15	0.64
F3D	I3D	0.52	88.96	+3.19	96.81	0.55	0.39	70.20	+3.30	87.17	0.56
F3D	R3D	0.33	90.29	+4.52	97.34	0.55	0.30	69.99	+3.09	87.22	0.55
GCN	I3D	0.36	92.15	+6.38	97.61	0.40	0.29	74.20	+7.30	90.02	0.47
GCN	R3D	0.37	91.49	+5.72	97.47	0.43	0.38	74.06	+7.16	89.30	0.47
GCN	F3D	0.48	91.49	+5.72	97.21	0.38	0.38	74.06	+7.16	89.51	0.43

Table 4.17: Test accuracy of two-stream networks on WLASL-100 and WLASL-2000 datasets

Streams		WLASL-100					WLASL-2000				
#1	#2	α^*	Top-1	I.O.B.	Top-5	Corr.	α^*	Top-1	I.O.B.	Top-5	Corr.
I3D	-	-	82.56	-	92.64	-	-	52.88	-	83.81	-
I3D	R3D	0.18	82.17	-0.39	95.35	0.52	0.49	56.22	+3.34	87.28	0.63
F3D	I3D	0.50	83.72	+1.16	95.35	0.39	0.25	55.49	+2.61	86.87	0.53
F3D	R3D	0.50	84.11	+1.55	94.96	0.45	0.28	55.49	+2.61	86.66	0.52
GCN	I3D	0.39	87.21	+4.65	95.35	0.36	0.41	60.46	+7.57	91.07	0.48
GCN	R3D	0.29	87.21	+4.65	95.35	0.38	0.38	60.39	+7.51	90.69	0.47
GCN	F3D	0.28	84.88	+2.33	94.19	0.31	0.52	58.69	+5.80	89.40	0.41

from the baseline model (% Error ↓) is calculated as follows:

$$\%\text{Error Reduction} = \frac{\text{baseline error rate} - \text{two-stream error rate}}{\text{baseline error rate}} \times 100 \tag{4.10}$$

where the error rate is defined as 100–(top-1 accuracy).

We include this measure in Table 4.15 (for AUTSL dataset) because the top-1 accuracies of the combined streams on AUTSL dataset are in the high 90s, so the reduction in error rate is more meaningful metric than percentage point improvement in accuracy (I.O.B.).

The error correlation (Corr.) column is used as a measure of diversity in the two-stream ensemble, which we calculate in two steps. First, the multi-class predictions for both models are converted into binary error vectors $e^{(k)} = \{e_i^{(k)}\}_{i=1}^{N}$, defined element-wise as follows:

$$e_i^{(k)} = \begin{cases} 1 & \text{if } \hat{y}_i^{(k)} \neq y_i, \\ 0 & \text{if } \hat{y}_i^{(k)} = y_i. \end{cases} \tag{4.11}$$

where k denotes the model index in the two streams ($k = 1$ for the first model, $k = 2$ for the second model), $\hat{y}_i^{(k)}$ is the predicted class label for instance i by the k-th model, y_i is the true class label for instance i, and $e_i^{(k)}$ indicates if the k-th model's prediction for instance i is incorrect (1) or correct (0).

Then, we apply Matthew's Correlation Coefficient (MCC) [103] to the binary error vectors $e^{(1)}$ and $e^{(2)}$ to measure the error correlation between the two streams as follows:

$$MCC = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.12}$$

where

- TP: Number of true positives (both models agree on errors).
- TN: Number of true negatives (both models agree on correct predictions).

- FP: Number of false positives (model 1 predicts correctly, model 2 predicts incorrectly).

- FN: Number of false negatives (model 1 predicts incorrectly, model 2 predicts correctly).

Note that MCC here is equivalent to Pearson's Correlation Coefficient for binary variables (Phi coefficient).

The results in Tables 4.15, 4.16, and 4.17 reveal the following findings.

- **Increased I.O.B. with Task Complexity**: As the learning task becomes more challenging, with a larger number of classes and fewer training samples per class, we observe slightly better improvement over baseline (I.O.B.). This trend can be seen in the increasing improvements from AUTSL to MSASL-100/WLASL-100 to MSASL-1000/WLASL-2000.

- **Skeleton Stream's Advantage in Accuracy Gains**: The largest accuracy improvement over the single-stream baseline (I3D) consistently occurs when combining a skeleton stream (GCN) with an RGB stream (I3D or R3D). While combining an optical flow stream (F3D) with an RGB stream improves top-1 accuracy by approximately $1-4$ percentage points, adding a skeleton stream to an RGB stream increases accuracy by around $4-7$ percentage points on the MSASL and WLASL datasets, and achieves a $41-53\%$ reduction in error rate on the AUTSL dataset.

- **Limited Benefit of Optical Flow Stream with RGB**: Although combinations like [F3D, I3D] or [F3D, R3D] show improvement over the single-stream baseline, they do not improve accuracy over the two-stream single-modality combination [I3D, R3D] by a meaningful margin. This suggests that the use of an optical flow stream with an RGB stream may not be justified, because it introduces additional computational overhead from the optical flow extractor without significant accuracy improvement over [I3D, R3D].

- **Influence of Error Correlation on Ensemble Accuracy**: The accuracy of a two-stream network depends on both the accuracy of the individual streams and the correlation of their errors. Two-stream networks with lower error correlation, such as RGB and skeleton combinations, exhibit more diversity in their predictions which increases the potential for improvement, as one stream can correct some of the false predictions made by the other. By contrast, streams of the same modality (e.g., I3D and R3D) tend to have higher error correlation, resulting in less diversity and an increased likelihood of similar misclassifications.

Based on the presented results, we recommend using [GCN, I3D] as the two-stream network for sign recognition since it achieves high accuracy relative to other combinations while minimizing computation (FLOPs), given that I3D is a lighter RGB model than R3D.

Quantitative and Qualitative Analysis

The improvement achieved by the two-stream ensemble (e.g., [GCN, I3D]) over the single-stream baseline (I3D) can be broken down into the following components.

1. **Exclusive Ensemble Gain**: Number of samples that are correctly classified only by the ensemble, but misclassified by both I3D and GCN.

$$\sum_{i=1}^{N} [\text{EnsembleCorrect}(i) \wedge \neg\text{I3DCorrect}(i) \wedge \neg\text{GCNCorrect}(i)] \tag{4.13}$$

2. **GCN-based Ensemble Gain**: Number of samples that are correctly classified by both GCN and the ensemble, but misclassified by I3D.

$$\sum_{i=1}^{N} [\text{EnsembleCorrect}(i) \wedge \text{GCNCorrect}(i) \wedge \neg\text{I3DCorrect}(i)] \tag{4.14}$$

3. **Ensemble Loss**: Number of samples that are correctly classified by I3D, but misclassified by both GCN and the ensemble.

$$\sum_{i=1}^{N} [\text{I3DCorrect}(i) \wedge \neg\text{GCNCorrect}(i) \wedge \neg\text{EnsembleCorrect}(i)] \tag{4.15}$$

where N is the total number of samples in the test set, and EnsembleCorrect(i), GCNCorrect(i), and I3DCorrect(i) return true if the test sample i is correctly classified by the respective model, and false otherwise. The Iverson bracket [ ] returns 1 if the condition inside it is true, and 0 otherwise.

The total ensemble gain is the sum of the two types of ensemble gains:

$$\text{Total Ensemble Gain} = \text{Exclusive Ensemble Gain} + \text{GCN based Ensemble Gain} \tag{4.16}$$

The net gain of the ensemble is calculated as follows:

$$\text{Ensemble Net Gain} = \text{Total Ensemble Gain–Ensemble Loss} \tag{4.17}$$

For convenience, we expressed the decomposition in Eqs. 4.13, 4.14, 4.15, and 4.16 using the [GCN, I3D] combination as an example. However, the same three-component decomposition applies to any two-stream network combination by replacing GCN with Stream1 and I3D with Stream2.

It is worth noting that if both streams classify a sample correctly, their ensemble will always classify it correctly, regardless of the fusion weighting. Therefore, such cases are not counted in the ensemble improvement decomposition.

The ensemble improvement decomposition can be further broken down by class, as shown in Figure 4.9. The top of Figure 4.9 illustrates the number of samples per class classified correctly only by I3D (cyan bars) or only by GCN (red bars), without the ensemble. The bottom of Figure 4.9 shows the ensemble's effect as follows.

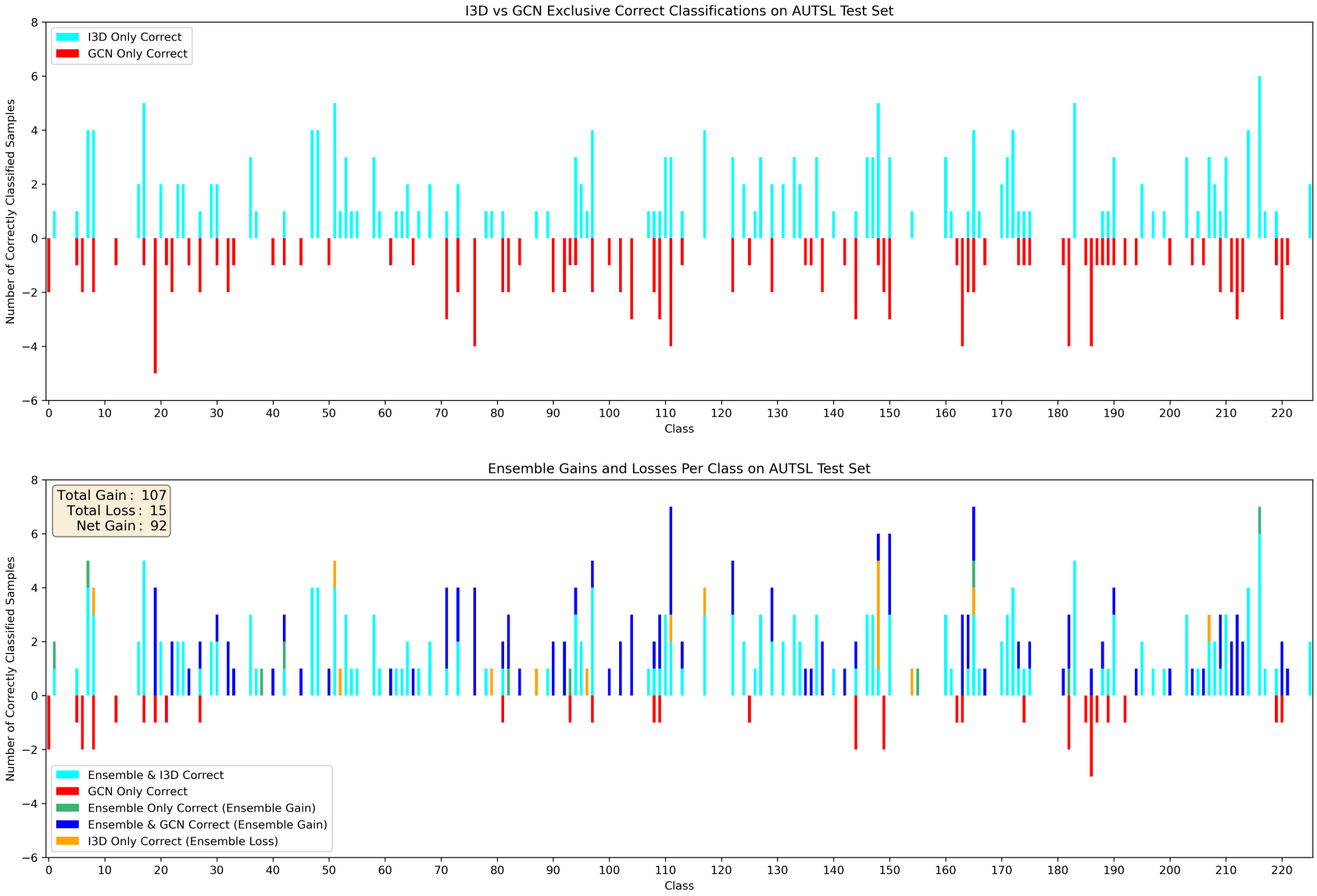


Figure 4.9: Decomposition of ensemble gains and losses per class on AUTSL dataset.

- **97 samples** are GCN-based ensemble gains. These samples were previously in the red bars in the top figure, correctly classified only by GCN, but are now correctly classified by the ensemble as well (dark blue bars).

- **10 samples** are exclusive ensemble gains. These samples were classified incorrectly by both I3D and GCN as single models but are now correctly classified by the ensemble (green bars).

- **15 samples** are ensemble losses. These samples are classified correctly by I3D but are now misclassified by the ensemble (orange bars).

The total net gain is $97 + 10\text{–}15 = 92$ samples out of 3,742 total samples in the AUTSL test set, which is a 2.46% improvement over the baseline, matching the I.O.B. value reported in Table 4.15 for [GCN, I3D].

Figure 4.9 shows that the maximum gain or loss per class did not exceed 4 samples. Although this is not statistically significant enough to conclude that the ensemble is particularly beneficial for certain classes of signs compared to the baseline, examining the new samples classified correctly or incorrectly by the ensemble from quantitative and qualitative perspectives can provide insight into the factors behind these outcomes. In the following analysis, we take a closer look at the ensemble gain in class #76 and the ensemble loss in class #148 on AUTSL dataset, where the [GCN, I3D] ensemble demonstrated its maximum gain and loss, respectively.

The left column of Figure 4.10 (first row) shows a sample of class #76 "see" that was correctly classified by the GCN stream, but misclassified by the I3D stream as class #22 "look". The signer holds up two fingers in "see" but one finger in "look." In this instance, the second finger was partially obscured by the index finger, leading the RGB stream (I3D) to incorrectly perceive the two fingers as one.

Figure 4.10: Challenging cases for I3D and GCN streams. Left column: the original sign. Right column: what the stream confused it with.

Table 4.18 demonstrates how the weighted ensemble classified this instance correctly from a numerical perspective. By weighting the GCN logit with $\alpha_{GCN} = 0.27$ and the I3D logit with $1 - \alpha_{GCN} = 0.73$ for both classes (#76 and #22), the resulting ensemble logit for class #76 (1.4921) is greater than that for class #22 (1.4266), making class #76 the top prediction of the ensemble.

Table 4.18: Examples of the ways the weighted combinations of logits affect the final prediction.

		I3D stream		**GCN stream**		**Ensemble** ($\alpha_{\text{GCN}} = 0.27$)	
		Class#	**Logit**	**Class#**	**Logit**	**Class#**	**Logit**
GCN-based Ensemble Gain (Class#76)	Top prediction	22	1.0268	76	9.5812	76	1.4921
	Contender	76	-1.4998	22	2.5075	22	1.4266
Ensemble Loss (Class#148)	Top prediction	148	0.1430	103	7.6864	103	1.9227
	Contender	103	-0.2091	148	3.2898	148	0.9926
Exclusive Ensemble Gain (Class#42)	Top prediction	136	-0.4694	93	7.2144	42	0.7019
	Contender	42	-0.5818	42	4.1726	136	0.2758
No Change (Class#6)	Top prediction	79	1.2066	79	7.5471	79	2.9186
	Contender	194	-1.0480	136	2.1738	194	-0.1818

A contrasting example is given in the second row of Figure 4.10, where a sample from class #148 "they" was classified correctly by the I3D stream but misclassified by the GCN stream as class #103 "pencil." Both classes have similar hand shape (index finger extended), but differ in their motion. The sign for "they" includes a 3D circular motion, while the sign for "pencil" involves a 2D left-right motion. Our skeleton representation includes vertical and horizontal coordinates but lacks a depth coordinate, making it challenging for the GCN stream to distinguish between circular and linear motions. Notably, the fact that the I3D model can differentiate between these two signs further supports that an optical flow stream is not required, even for such subtle distinctions.

Although this instance was correctly classified by I3D, it was misclassified by the ensemble due to the weighting of logits from the two streams, as shown in Table 4.18. The difference

between the GCN logits for classes #103 and #148 is significantly larger than that of I3D, so even a 0.27 weighting of the GCN logit caused the resulting ensemble logit for the incorrect class #103 (1.9227) to be higher than that for the correct class #148 (0.9926). While adjusting the weight α_{GCN} might seem like a potential solution for this specific instance, it would be suboptimal for many other cases and would decrease the overall accuracy of the ensemble. Another way to address this issue is to add a depth coordinate to the skeleton representation. However, aside from a few cases, we have not observed a meaningful increase in overall accuracy by adding a depth coordinate.

Exclusive ensemble gain can occur, though not necessarily, when each stream has a different incorrect top prediction for a sample, but they both predict the correct class as a close contender. If the combined weighted logits of the contender exceed that of both stream's top prediction, the sample will be classified correctly by the ensemble as occurred in class #42 "child" (Table 4.18).

The most challenging sign for the ensemble to classify correctly in the AUTSL dataset belongs to class #6 "heavy", which had the lowest recall of any class: 38.5% on the validation set and 47.1% on the test set. Class #6 "heavy" was frequently confused with class #79 "lightweight", where 53.8% of the validation samples and 35.3% of the test samples for "heavy" were misclassified as "lightweight". The third row of Figure 4.10 illustrates this confusion, where a sample for "heavy" was misclassified by I3D, GCN, and their ensemble as a "lightweight" sign.

The distinction between these two signs is challenging because they share the same hand shape and motion, with both hands lifted up as if carrying weight, but "heavy" is signed with more intensity in facial expression to convey heavyweight. This confusion might be mitigated in continuous sign recognition with word-context awareness. Another way to address this in isolated sign recognition would be to add a stream with attention to facial expressions. However, since signs that are identical in all aspects except for subtle facial expressions are rare, this added complexity may not be justified.

4.7 Results of Three-Stream Networks

There are ${}^4C_3 = 4$ possible combinations to form a three-stream network from the four models: I3D, R3D, F3D, and GCN. Equivalently, these combinations can also be formed by starting from the four-stream network, including all four models, and choosing one stream to remove each time (${}^4C_1 = 4$) while keeping the other three in the network.

The test results of all three-stream combinations on AUTSL, MSASL, and WLASL datasets are reported in Tables 4.19, 4.20, and 4.21, respectively. We define two types of Improvement Over Baseline (I.O.B.) to evaluate the performance of each three-stream combination.

- **I.O.B.2**: The difference in top-1 accuracy relative to the two-stream baseline [I3D, GCN].
- **I.O.B.4**: The difference in top-1 accuracy relative to the full four-stream network [I3D, R3D, F3D, GCN].

We selected [I3D, GCN] (first row) as our two-stream baseline due to its higher accuracy and computational efficiency compared to other two-stream combinations, as mentioned previously. The accuracy of the four-stream combination [I3D, R3D, F3D, GCN] (second row) is reported based on its optimal weight combination, determined using grid search as outlined in Algorithm 1 but with an additional fusion weight for the fourth stream.

The following insights can be drawn from the results in Tables 4.19, 4.20, and 4.21.

- **Importance of the Skeleton Stream**: Removing the skeleton stream (GCN) from the four-stream network consistently results in a more significant drop in accuracy than removing any other steam. For instance, in MSASL and WLASL, the three-stream combination [I3D, R3D, F3D] has approximately $3-4$ percentage points lower accuracy than three-stream combinations that include GCN. Furthermore, [I3D, R3D, F3D] network consistently performs worse than the two-stream baseline [I3D, GCN].

Table 4.19: Test accuracy of multi-stream networks on AUTSL dataset. I.O.B.2 and I.O.B.4 denote the difference in top-1 accuracy relative to the two-stream baseline (first row) and the four-stream network (second row), respectively.

Streams				AUTSL			
I3D	R3D	F3D	GCN	Top-1	I.O.B.2	I.O.B.4	Top-5
✓	✗	✗	✓	96.50	-	-0.80	99.71
✓	✓	✓	✓	97.30	+0.80	-	99.79
✗	✓	✓	✓	97.30	+0.80	0.00	99.79
✓	✗	✓	✓	96.58	+0.08	-0.72	99.73
✓	✓	✗	✓	97.01	+0.51	-0.29	99.87
✓	✓	✓	✗	95.54	-0.96	-1.76	99.68

Table 4.20: Test accuracy of multi-stream networks on MSASL-100 and MSASL-1000 datasets

Streams				MSASL-100				MSASL-1000			
I3D	R3D	F3D	GCN	Top-1	I.O.B.2	I.O.B.4	Top-5	Top-1	I.O.B.2	I.O.B.4	Top-5
✓	✗	✗	✓	92.15	-	-0.40	97.61	74.20	-	-1.91	90.02
✓	✓	✓	✓	92.55	+0.40	-	97.87	76.11	+1.91	-	91.03
✗	✓	✓	✓	92.55	+0.40	0.00	97.87	75.80	+1.60	-0.31	90.60
✓	✗	✓	✓	92.82	+0.67	+0.27	97.74	75.41	+1.21	-0.70	90.77
✓	✓	✗	✓	92.82	+0.67	+0.27	98.01	75.77	+1.57	-0.34	90.48
✓	✓	✓	✗	90.16	-1.99	-2.39	97.07	72.73	-1.47	-3.38	88.04

Table 4.21: Test accuracy of multi-stream networks on WLASL-100 and WLASL-2000 datasets

Streams				WLASL-100				WLASL-2000			
I3D	R3D	F3D	GCN	Top-1	I.O.B.2	I.O.B.4	Top-5	Top-1	I.O.B.2	I.O.B.4	Top-5
✓	✗	✗	✓	87.21	-	-1.55	95.35	60.46	-	-1.74	91.07
✓	✓	✓	✓	88.76	+1.55	-	95.74	62.20	+1.74	-	92.22
✗	✓	✓	✓	88.76	+1.55	0.00	95.74	61.26	+0.80	-0.94	91.66
✓	✗	✓	✓	87.98	+0.77	-0.78	95.35	61.33	+0.87	-0.87	91.66
✓	✓	✗	✓	87.98	+0.77	-0.78	94.96	61.71	+1.25	-0.49	92.08
✓	✓	✓	✗	84.50	-2.71	-4.26	95.74	57.51	-2.95	-4.69	88.78

This demonstrates the importance of the skeleton stream in any two- or three-stream combinations.

- **Minimal Impact of the Optical Flow Stream**: Removing the optical flow stream (F3D) from the four-stream network has a similar negligible effect on the accuracy as removing I3D or R3D (less than 1 percentage point drop). This suggests that the optical flow stream, whose predictions are more correlated with RGB streams, can be replaced by another RGB stream, such as I3D or R3D, without a meaningful drop in accuracy.

- **Diminishing Returns with Additional Streams**: The maximum gain of two-stream networks over the single-stream baseline in Tables 4.15, 4.16, and 4.17 was about 7.5 percentage points, which is much more significant improvement than the maximum gain of 1.6 percentage points for three-stream networks over the two-stream baseline as shown in Tables 4.19, 4.20, and 4.21. Moreover, the maximum gain of the four-stream network [I3D, R3D, F3D, GCN] over any three-stream combination that includes GCN was less than 1 percentage point in top-1 accuracy and almost no improvement in top-5 accuracy. This points to diminishing returns, as adding more streams increases the computational cost but results in progressively smaller gains in accuracy.

Based on these results, the two-stream network [I3D, GCN] has shown to be a strong baseline, as all three-stream combinations across different datasets showed only marginal improvements over it. However, if slightly higher accuracy is required, [I3D, R3D, GCN] is preferable to [I3D, F3D, GCN] or [R3D, F3D, GCN] due to the added computational overhead associated with optical flow extraction.

4.8 Comparison with State-of-the-Art Methods

In this section, we compare the performance of our multi-stream networks against two state-of-the-art (SotA) methods: SignBERT+ [17] and SAM-SLR-v2 [88]. Table 4.22 summarizes the test accuracy of various models on the AUTSL, MSASL, and WLASL datasets. The publications on these methods did not provide information about their computational costs. Therefore, we estimate certain components of these methods to compare their computational efficiency. Table 4.23 presents the GFLOPs count for each modality in each method, determined by using the FlopCountAnalysis function in the fvcore library [80] and confirmed with the ptflops tool [48]. Since our objective is to maximize accuracy while minimizing computational cost, we use [GCN, I3D] as our two-stream network and [GCN, I3D, R3D] as our three-stream network.

In each of the following sections (4.8.1 and 4.8.2), we first provide a summary of the SotA method followed by a discussion of the accuracy and computational efficiency compared to our models.

4.8.1 SignBERT

SignBERT [105] is a self-supervised pretraining framework designed for sign language tasks using hand pose information. It incorporates ideas from BERT's architecture and its pretraining strategies [66] but treats hand poses as visual tokens analogous to textual tokens in BERT. SignBERT first performs self-supervised pretraining by masking some hand poses in the input, passing them through a transformer encoder, and reconstructing them using a hand-model aware decoder. The model can then be finetuned on a downstream task such as sign language recognition (SLR) by replacing the decoder with a prediction head. SignBERT+ [17] extends the original framework to two additional downstream tasks: continuous SLR and sign language translation (SLT), along with other techniques to enhance performance.

Table 4.22: Test accuracy comparison of SotA methods on sign language benchmarks. Accuracy with (*) indicates per-class accuracy.

Method	AUTSL		MSASL-100		MSASL-1000		WLASL-100		WLASL-2000	
	Top-1	**Top-5**	**Top-1**	**Top-5**	**Top-1**	**Top-5**	**Top-1**	**Top-5**	**Top-1**	**Top-5**
AUTSL [28]	49.22	75.78	-	-	-	-	-	-	-	-
I3D [26, 27]	-	-	81.76*	95.16*	57.69*	81.08*	65.89	84.11	32.48	57.31
BSL [104]	-	-	-	-	64.71	85.59	-	-	46.82	79.36
SAM-SLR [79]	97.51	100	-	-	-	-	-	-	58.73	91.46
SAM-SLR-v2 [88]	**98.00**	**100**	-	-	-	-	-	-	59.39	91.48
SignBERT+ (H) [17]	-	-	84.94	95.77	62.42	83.49	79.84	92.64	48.85	82.48
SignBERT+ (H+R) [17]	-	-	90.75	97.75	73.71	90.12	84.11	**96.51**	55.59	89.37
Ours [GCN, I3D]	96.50	99.71	92.15	97.61	74.20	90.02	87.21	95.35	60.46	91.07
Ours [GCN, I3D, R3D]	97.01	99.87	**92.82**	**98.01**	**75.77**	**90.48**	**87.98**	94.96	**61.71**	**92.08**

Table 4.23: Estimated computational cost (GFLOPs/view) for SotA methods. Lower is better. S_1: MediaPipe Skeleton, S_2: HRNet Skeleton, F: TV-L1 Optical Flow, T: Transformer Encoder

Method	AUTSL	MSASL/WLASL
SAM-SLRv2 [88]	RGB: 424.7×5 Flow: $424.7 \times 5 + F$ Skeleton: $2.3 \times 4 + S_2$ Total: $4{,}256.2 + F + S_2$	RGB: 424.7×5 Flow: $424.7 \times 5 + F$ Skeleton: $2.3 \times 4 + S_2$ Total: $4{,}256.2 + F + S_2$
SignBERT+ (H+R) [17]	-	RGB: 145.4 Skeleton: $2.3 + S_2 + T$ Total: $147.7 + S_2 + T$
Ours [GCN, I3D]	RGB: 227.2 Skeleton: $2.3 + S_1$ **Total: 229.5 + S_1**	RGB: 145.4 Skeleton: $2.3 + S_1$ **Total: 147.7 + S_1**
Ours [GCN, I3D, R3D]	RGB: $227.2 + 1330.3$ Skeleton: $2.3 + S_1$ **Total: 1,559.8 + S_1**	RGB: $145.4 + 851.4$ Skeleton: $2.3 + S_1$ **Total: 999.1 + S_1**

In Table 4.22, SignBERT+ (H) refers to the core transformer model using hand skeleton input, while SignBERT+ (H+R) denotes an ensemble of both the hand skeleton (H) and RGB model (R) as described in [17]. Our [GCN, I3D] model differs from SignBERT+ (H+R) in two key ways.

1. For the MSASL and WLASL datasets, the individual streams in our approach are pre-trained coarse-to-fine on Kinetics then AUTSL in a **supervised** manner. In contrast, SignBERT+ employs **self-supervised** pretraining on a collection of sign language datasets, comprising a large volume of 230,246 videos [17].

2. SignBERT+ (H) is designed to learn signs based solely on hand joints, while our skeleton model incorporates both hand joints and other landmarks such as face, elbows, and shoulders. Notably, our AUTSL-pretrained GCN model alone (Table 4.14) achieves higher accuracy than SignBERT+ (H) on WLASL2000 (52.57 vs. 48.85) and a comparable accuracy on MSASL1000 (63.93 vs. 62.42). This suggests that including other body joints provides valuable cues for recognizing signs.

Compared to the accuracy of SignBERT+ (H+R) (Table 4.22), our two-stream network [GCN, I3D] achieves +4.41 and +3.1 percentage points improvement on WLASL2000 and WLASL100, respectively, and a comparable accuracy with a slight increase of +0.5 and +1.4 points on MSASL1000 and MSASL100, respectively. The relatively higher gains on WLASL compared to MSASL indicate the effectiveness of our training method for datasets with low training sample density, such as WLASL.

The estimated computational costs of SignBERT+ (H+R) and our two-stream network are provided in Table 4.23. Both methods have similar inference costs for the RGB model, at 145.4 GFLOPs per view (i.e., 64 frames of size 256×256). However, the computational cost of the skeleton models differs in two key aspects.

- The skeleton in SignBERT+ is extracted from frames using HRNet [33], while we use MediaPipe [34, 35] for skeleton extraction. HRNet provides higher accuracy for

body and hand pose estimation than MediaPipe but is more computationally intensive. MediaPipe, on the other hand, can run in real time on a smartphone. On a desktop CPU, our measurements show MediaPipe achieves a processing rate of 20 frames per second for hand pose extraction and 10 frames per second for body and hand pose extraction, compared to HRNet's processing speed of 1.3 frames per second on CPU. Since no documentation is available on the FLOPs count for MediaPipe, we denote the FLOPs count as S_1 for MediaPipe and S_2 for HRNet. Given that MediaPipe's processing speed on CPU is an order of magnitude faster than HRNet, it is reasonable to assume $S_1 < S_2$.

- The skeleton data in our approach is fed into ST-GCN for classification, whereas SignBERT+ embeds the hand skeleton using spectral-based GCN [106] and then processes the embedding with a transformer architecture. Spectral-based GCN is more computationally intensive than ST-GCN as it requires eigen-decomposition of the graph Laplacian. However, this cost can be mitigated by using a first-order approximation of the graph Laplacian [106]. For simplicity, we assume a comparable FLOPs count of 2.3 GFLOPs for both types of GCNs. Nevertheless, SignBERT+ has the additional computational cost of the transformer architecture, denoted by T.

To summarize, the total computational cost of our two-stream [GCN, I3D] approach is $(147.7 + S_1)$ GFLOPs/view, and for SignBERT+ it is $(147.7 + S_2 + T)$ GFLOPs/view. Since $S_2 > S_1$ and $T >> 0$, our approach is more computationally efficient while demonstrating better accuracy.

4.8.2 SAM-SLR

Skeleton Aware Multi-modal Sign Language Recognition (SAM-SLR) [79] uses six streams from five different modalities: RGB, Depth, RGB Flow, Depth Flow, and Skeleton, to achieve a higher sign recognition rate compared to previous methods. For RGB, Depth, and their corresponding optical flow, the ResNet2+1D [4] backbone is used for each stream. For skele-

ton modality, they propose SL-GCN model for skeleton key points and SSTCN model for skeleton features. The optical flow is extracted using the TV-L1 [107] algorithm, while skeleton key points and features are extracted by HRNet [33]. In SAM-SLR-v2 [88], they propose Global Ensemble Model (GEM), a multi-layer network to learn the weighting of each stream, resulting in slightly better accuracy.

SAM-SLR-v2 reported results on two tracks: the RGB-D track, including all the aforementioned modalities, and the RGB track, which excludes depth and depth flow. For a fair comparison, we evaluate the accuracy and computational cost of our approach against the RGB track, as both operate on RGB videos. Nevertheless, adding depth and depth flow in SAM-SLR-v2 resulted in an insignificant improvement of just +0.10 to the top-1 accuracy of 98.00% as reported on the AUTSL dataset [88].

Compared to SAM-SLR-v2, our three-stream network [GCN, I3D, R3D] achieves a slightly lower top-1 accuracy of 97.01% on AUTSL. However, on the WLASL1000 dataset, our two- and three-stream networks outperform SAM-SLR-v2 with improvements of +1.07 and +2.32 percentage points, respectively. It is worth noting that our approach achieves such accuracy while requiring significantly less computational cost, as demonstrated in Table 4.23. The comparison of computational efficiency is detailed as follows.

- **RGB**: The ResNet2+1D streams used for RGB and RGB Flow of SAM-SLR-v2 perform 2D spatial convolution followed by 1D temporal convolution in each layer. While the FLOPs count of ResNet2+1D is comparable to that of ResNet3D (i.e. R3D) used in our approach, our R3D utilizes 2.75x less memory during the forward pass, as we measured using torchsummary [108]. Additionally, although our approach uses a larger view size of 320×320 pixels for inference on AUTSL compared to 256×256 in SAM-SLR-v2, the latter performs inference with 5 temporal crops, each with 32 frames.

  In total, their FLOPs count for RGB and Flow is estimated as $2 \times 424.7 \times 5 + F = 4,247 + F$ GFLOPs, where F denotes the FLOPs count of the TV-L1 algorithm used by [107] to extract optical flow. In contrast, our two RGB streams, I3D and R3D, have

a total FLOPs count of $227.2 + 1330.3 = 1557.5$ GFLOPs for AUTSL. For MSASL and WLASL, the total is reduced to 996.8 GFLOPs due to the use of a smaller view size of 256×256 pixels.

- **Skeleton**: Similar to SignBERT+, SAM-SLR-v2 employs HRNet for extracting skeleton key points and features, while our approach utilizes the faster MediaPipe, as discussed in Section 4.8.1. SAM-SLR-v2 uses four streams of SL-GCN for joint, bone, joint motion, and bone motion, as well as one stream of SSTCN for skeleton features. In contrast, we use a single stream of ST-GCN with joint and bone information combined as channels instead of separate streams.

  To simplify the calculations without affecting the outcome, we use the same estimation as in Section 4.8.1 of 2.3 GFLOPs for both SL-GCN and ST-GCN and disregard the computational cost of SSTCN. Therefore, the total FLOPs count for the skeleton modality is estimated as $2.3 \times 4 + S_2 = 9.2 + S_2$ GFLOPs/view for SAM-SLR-v2, and $(2.3 + S_1)$ GFLOPs/view for our approach, where S_1 and S_2 represent the FLOPs counts of MediaPipe and HRNet, respectively.

The total computational cost of our three-stream network [GCN, I3D, R3D] is $(1,559.8 + S_1)$ GFLOPs/view for AUTSL and $(999.1 + S_1)$ GFLOPs/view for MSASL and WLASL, while SAM-SLR-v2 consumes $(4,256.2 + F + S_2)$ GFLOPs/view. Given that $S_2 > S_1$ and $F >> 0$, our three-stream approach [GCN, I3D, R3D] reduces the FLOPs count in the second stage by at least 76% while achieving state-of-the-art accuracy on the WLASL and MSASL datasets.

Chapter 5

Conclusions

This dissertation addressed the problem of isolated sign language recognition in videos, a fine-grained classification problem characterized by specific hand shapes, orientations, and motion patterns. Motivated by the need to maximize recognition accuracy while minimizing computational cost, we constructed, trained, evaluated, and analyzed the performance of single- and multi-stream networks with RGB, optical flow, and skeleton inputs. In this chapter, we provide a summary of our findings in Section 5.1 and directions for future research in Section 5.2.

5.1 Summary of Findings

In our evaluations, the significance of our results is determined by consistent improvements across multiple datasets. This eliminates the need for repeated runs on the same dataset since improvements across different datasets provide stronger evidence through independent confirmations. Additionally, the three sign language datasets (AUTSL, MSASL1000, and WLASL2000) have relatively large test sets (3,000–4,000 videos), leading to stable results with low variance when training and testing are repeated on the same dataset.

Our experiments on slow fusion, in which temporal features across frames are learned in every layer using 3D convolutions, consistently demonstrate significant accuracy improvement over late fusion methods, such as averaging or using a recurrent layer. This improvement is more pronounced than what has been documented in action recognition literature. Moreover, a shallower 3D convolutional network of 18 layers significantly outperforms a deeper 2D convolutional network with 50 layers, highlighting the importance of slow fusion in sign language recognition. Notably, signs where motion is the primary discriminative feature are better classified with R3D than R2D.

The primary value of optical flow lies in its ability to segment relevant regions for sign recognition rather than learning of motion patterns, as previously assumed. We demonstrated this in two ways.

i. No statistically significant difference was observed in the accuracy of motion-sensitive classes between the RGB model (I3D) and the optical flow model (F3D)

ii. Removing direction from optical flow retains the same accuracy, and replacing optical flow with a binary mask of its magnitude causes only a minor drop in accuracy ($< 3\%$). These findings indicate that learning motion patterns useful for sign recognition is primarily attributed to the slow fusion mechanism in the network, rather than the optical flow input itself.

Two-stream networks with RGB and optical flow inputs have been extensively adopted in action and sign recognition literature. However, these methods often lack a reference comparison to the baseline of an equivalent two-stream network where the optical flow stream is substituted with another RGB stream. To address this gap, we evaluated all possible two-stream combinations of RGB models (I3D and R3D), an optical flow model (F3D), and a skeleton model (GCN). The results show that the accuracy of two-stream networks [F3D, R3D] and [F3D, GCN] is comparable to that of [I3D, R3D] and [I3D, GCN], respectively. These findings suggest that optical flow is replaceable without a loss of accuracy, leading to significant savings in computation and energy.

The structured representation of skeleton data as a spatiotemporal graph, with sparsely connected nodes representing hand and upper body joints, provides a better inductive bias for sign language recognition than a flattened representation used with a recurrent neural network or a transformer encoder. Our results show that the temporal convolutions in ST-GCN layers are as critical for the skeleton modality as the slow fusion of temporal features is in 3D convolutional networks for RGB. Both achieve significant improvement in accuracy compared to models using only spatial convolutions. To improve the interpretability of ST-

GCN, we proposed a method to visualize the importance of each edge in sign recognition based on the adjacency matrix and edge importance weighting.

Skeleton-based SLR with GCN outperforms optical-flow-based SLR with F3D while consuming less computation in both the first and second stages. This superiority is demonstrated in several ways.

i. As a single stream, GCN achieves higher per-instance and per-class accuracy across the datasets than F3D.

ii. In two-stream networks, adding the skeleton stream to an RGB stream consistently improves accuracy compared to ensembles of RGB and optical flow streams or two RGB streams, due to the lower error correlation between RGB and skeleton predictions, which enhances ensemble diversity.

iii. In multi-stream networks combining RGB, optical flow, and skeleton inputs, removing the skeleton stream results in a larger drop in accuracy than removing any other stream, while removing optical flow has an insignificant effect ($< 1\%$) on ensemble accuracy.

To address the limitation of insufficient training samples, we applied a coarse-to-fine training strategy, where the model is first trained on a general action recognition dataset, then on a different sign language dataset, and finally on the target dataset. Our ablation study shows that this strategy is most effective when the datasets used in transfer learning possess both large mass (total number of training samples) and high density (number of training samples per class). Our experiments demonstrate that features learned from AUTSL, a Turkish sign language dataset, are effective when transferred to two American sign language datasets, MSASL and WLASL. State-of-the-art methods often apply transfer learning to a single model in the ensemble. In contrast, our results show that coarse-to-fine training benefits all models—with RGB, optical flow, or skeleton inputs—significantly improving their individual accuracies.

In multi-stream networks for sign recognition, state-of-the-art methods often rely on weighted combinations of stream outputs, either set manually or learned through a neural network. However, how these outputs are weighted has a noticeable effect on the ensemble accuracy, and systematic exploration of stream fusion methods for sign recognition has been limited. To address this, we investigated both pre-classification and post-classification fusion methods.

In post-classification fusion, we proposed grid search and gradient-based optimization methods to determine the optimal weighting of stream outputs. In pre-classification fusion, we trained streams simultaneously to learn a non-linear fusion of their output features and incorporated auxiliary classifiers to enhance accuracy. Our results show that learning fusion weights per class or group of classes did not improve the accuracy compared to a single-weight fusion method. Similarly, the non-linear fusion of features before the classification layer did not outperform the grid search baseline.

Our extensive evaluations of multi-stream networks demonstrate that the [I3D, GCN] combination provides the best trade-off between accuracy and computational efficiency. Adding additional streams results in diminishing returns, with progressively smaller accuracy gains and increased computational cost. Our ensemble improvement decomposition reveals no class-specific advantage for a specific stream. Instead, each stream contributes to the ensemble's accuracy at an instance level.

Our two-stream network [I3D, GCN] achieves new state-of-the-art performance on the MSASL and WLASL datasets while consuming significantly fewer FLOPs than other SotA methods in both the first and second stages. This success is attributed to a combination of factors, including coarse-to-fine transfer learning across all streams, optimal stream fusion weighting, and disciplined hyperparameter tuning of the learning rate, schedule, training epochs, batch size, and regularization. In particular, the one-cycle learning schedule and a small batch size were important for improving the accuracy of individual streams on MSASL and WLASL.

5.2 Future Work

The ultimate goal of automated sign language recognition is to improve convenience and accessibility in the daily lives of deaf individuals. This research advances the field towards that goal by providing theoretical insights and experimental results on the best methods and strategies for improving recognition rate while maintaining computational efficiency. Nevertheless, there are limitations and opportunities for further improvements, which we plan to address in future work, as outlined below.

Although our two-stream network [I3D, GCN] is more computationally efficient compared to other SotA methods, running the network as it is on mobile devices is challenging due to the high computational demands of I3D (145.4 GFLOP per 64-frame input with 256x256 spatial resolution). This challenge could be addressed in several ways.

i. Replacing I3D with the 3D version of resource-efficient, mobile-optimized architectures such as MobileNet, SqueezeNet, or ShuffleNet [109].

ii. Replacing I3D with Temporal Shift Module (TSM) to enable temporal modeling without heavy 3D convolutions [110].

iii. Using knowledge distillation to train a smaller student model to mimic the outputs of a larger teacher model [111].

iv. Applying model quantization to reduce the model size and increase inference speed by relying on integer arithmetic [112].

Future work should evaluate to what extent these optimizations affect the model's accuracy for sign recognition.

Additionally, the accuracy of the skeleton stream can be improved in multiple ways.

i. Pretraining GCN on the Kinetics dataset, followed by the AUTSL dataset, instead of AUTSL only, could yield significant improvements, similar to RGB streams.

ii. The skeleton extraction module (MediaPipe) produces missed hand detections in many frames (false negatives), which could be mitigated by conditioning hand detection on pose detection, as the latter has a relatively low false-negative rate.

iii. Attention-based or spectral-based graph models could be added as a second or third stream to GCN or [I3D, GCN], respectively. Fortunately, incorporating additional graph models in the second stage does not substantially increase the computational cost, as the bulk of computations occurs in the first stage (skeleton extraction module).

While our approach achieves over 90% top-5 accuracy on all isolated sign datasets, a significant hurdle to achieving over 90% top-1 accuracy on MSASL-1000 and WLASL-2000 is the low number of training videos for most signs. This becomes apparent when compared to our 96.5% top-1 and 99.7% top-5 accuracy on the AUTSL dataset, which contains an order of magnitude more training samples per class. This hurdle could potentially be mitigated in continuous sign language recognition, where each sign is performed within the context of a sequence, thus increasing the likelihood of correct identification. However, continuous SLR introduces additional challenges, such as sign localization and handling sign language grammar, which could be addressed by adapting successful methods from speech recognition and natural language processing.

Building on the insights and strategies presented in this dissertation, future work will explore extending our methods to continuous sign language understanding, paving the way for real-time communication systems that empower deaf individuals and help overcome communication barriers.

Bibliography

[1] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 4489–4497, 2015.

[2] J. Carreira and A. Zisserman, "Quo Vadis, action recognition? A new model and the kinetics dataset," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 4724–4733, 2017.

[3] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy, "Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11219 LNCS, pp. 318–335, 2018.

[4] D. Tran, H. Wang, L. Torresani, J. Ray, Y. Lecun, and M. Paluri, "A Closer Look at Spatiotemporal Convolutions for Action Recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 6450–6459, 2018.

[5] K. Soomro, "UCF101: A dataset of 101 human actions classes from videos in the wild," *arXiv preprint arXiv:1212.0402*, 2012.

[6] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "HMDB: A large video database for human motion recognition," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2556–2563, 2011.

[7] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. F. Li, "Large-scale video classification with convolutional neural networks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[8] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev *et al.*, "The kinetics human action video dataset," *arXiv preprint arXiv:1705.06950*, 2017.

[9] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local Neural Networks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 7794–7803, 2018.

[10] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional Two-Stream Network Fusion for Video Action Recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, no. i, pp. 1933–1941, 2016.

[11] J. Y. H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 4694–4702, 2015.

[12] R. Christoph and F. A. Pinz, "Spatiotemporal residual networks for video action recognition," *Advances in neural information processing systems*, vol. 2, pp. 3468–3476, 2016.

[13] J. Wan, C. Lin, L. Wen, Y. Li, Q. Miao, S. Escalera, G. Anbarjafari, I. Guyon, G. Guo, and S. Z. Li, "ChaLearn Looking at People: IsoGD and ConGD Large-Scale RGB-D Gesture Recognition," *IEEE Transactions on Cybernetics*, vol. 14, no. 8, pp. 1–12, 2020.

[14] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *Advances in Neural Information Processing Systems*, vol. 1, no. January, pp. 568–576, 2014.

[15] R. Adam, "WFD-WASLI Frequently Asked Questions about International Sign," [Online]. Available: https://wfdeaf.org/news/resources/faq-international-sign/, 2019, accessed: 2022-06-23.

[16] WHO, "Deafness and hearing loss," [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss, accessed: 2022-06-23.

[17] H. Hu, W. Zhao, W. Zhou, and H. Li, "Signbert+: Hand-model-aware self-supervised pre-training for sign language understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 11 221–11 239, 2023.

[18] H. Brashear, T. Starner, P. Lukowicz, and H. Junker, "Using multiple sensors for mobile sign language recognition," in *Seventh IEEE International Symposium on Wearable Computers, 2003. Proceedings.* IEEE Computer Society, 2003, pp. 45–45.

[19] A. Kuznetsova, L. Leal-Taixé, and B. Rosenhahn, "Real-time sign language recognition using a consumer depth camera," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 83–90, 2013.

[20] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 4645–4653, 2017.

[21] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton, and P. Presti, "American sign language recognition with the kinect," *ICMI'11 - Proceedings of the 2011 ACM International Conference on Multimodal Interaction*, pp. 279–286, 2011.

[22] R. Y. Wang and J. Popović, "Real-time hand-tracking with a color glove," *ACM Transactions on Graphics*, vol. 28, no. 3, pp. 1–8, 2009.

[23] V. Athitsos, C. Neidle, S. Sclaroff, J. Nash, A. Stefan, Q. Yuan, and A. Thangali, "The American Sign Language Lexicon Video Dataset," *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, no. June, 2008.

[24] X. Chai, H. Wang, and X. Chen, "The devisign large vocabulary of chinese sign language database and baseline evaluations," in *Technical Report VIPL-TR-14-SLR-001. Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS)*. Institute of Computing Technology, 2014.

[25] J. Huang, W. Zhou, H. Li, and W. Li, "Attention-based 3D-CNNs for large-vocabulary sign language recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 9, pp. 2822–2832, 2018.

[26] D. Li, C. R. Opazo, X. Yu, and H. Li, "Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison," *Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020*, pp. 1448–1458, 2020.

[27] H. R. V. Joze and O. Koller, "MS-ASL: A large-scale data set and benchmark for understanding American sign language," *30th British Machine Vision Conference 2019, BMVC 2019*, 2020.

[28] O. M. Sincan and H. Y. Keles, "AUTSL: A Large Scale Multi-modal Turkish Sign Language Dataset and Baseline Methods," *IEEE Access*, vol. 8, pp. 181 340–181 355, 2020.

[29] D. Li, C. R. Opazo, X. Yu, and H. Li, "WLASL (GitHub repository)," [Online]. Available: https://github.com/dxli94/WLASL, 2020, accessed: 2022-05-11.

[30] J. Redmon, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[31] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 815–823, 2015.

[32] M. S. A. College, "The 5 parameters of ASL," [Online]. Available: https://www.mtsac.edu/llc/passportrewards/languagepartners/5ParametersofASL.pdf, accessed: 2024-12-04.

[33] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang *et al.*, "Deep high-resolution representation learning for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 10, pp. 3349–3364, 2020.

[34] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee *et al.*, "Mediapipe: A framework for building perception pipelines," *arXiv preprint arXiv:1906.08172*, 2019.

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

[36] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems*, vol. 25, 2012.

[39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Ieee, 2009, pp. 248–255.

[40] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, "Action recognition by dense trajectories," in *Cvpr 2011*, 2011, pp. 3169–3176.

[41] H. Wang and C. Schmid, "Action recognition with improved trajectories," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3551–3558, 2013.

[42] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao, "Towards good practices for very deep two-stream convnets." *arXiv preprint arXiv:1507.02159*, 2015.

[43] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 677–691, 2017.

[44] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14, 2015.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 770–778, 2016.

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 1–9, 2015.

[47] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, pp. 448–456, 2015.

[48] V. Sovrasov and contributors, "Ptflops: Flops counting tool for neural networks in pytorch framework," [Online]. Available: https://github.com/sovrasov/flops-counter.pytorch, 2019, accessed: 2024-11-17.

[49] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-Octob, pp. 6201–6210, 2019.

[50] P. Turaga, R. Chellappa, and A. Veeraraghavan, "Advances in Video-Based Human Activity Analysis: Challenges and Approaches," in *Advances in Computers*. Elsevier, Jan. 2010, vol. 80, pp. 237–290.

[51] B. K. Horn and B. G. Schunck, "Determining Optical Flow," *Techniques and Applications of Image Understanding*, vol. 0281, pp. 319–331, 1981.

[52] G. Farneb, "Two-Frame Motion Estimation Based on Polynomial Expansion," *Lecture Notes in Computer Science*, vol. 2749, no. 1, pp. 363–370, 2003.

[53] C. Zach, T. Pock, and H. Bischof, "A Duality Based Approach for Realtime TV-L1 Optical Flow," *Pattern Recognition*, vol. 4713, pp. 214–223, 2007.

[54] T. Brox, N. Papenberg, and J. Weickert, "High Accuracy Optical Flow Estimation based on warping - presentation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3024, no. May, pp. 25–36, 2014.

[55] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "FlowNet 2.0: Evolution of optical flow estimation with deep networks," *Proceedings - 30th IEEE*

Conference on Computer Vision and Pattern Recognition, CVPR 2017, vol. 2017-Janua, pp. 1647–1655, 2017.

[56] Z. Teed and J. Deng, "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow (Extended Abstract)," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 4839–4843, 2021.

[57] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. van Gool, "Temporal segment networks: Towards good practices for deep action recognition," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9912 LNCS, pp. 20–36, 2016.

[58] A. M. Derrington and P. Lennie, "Spatial and temporal contrast sensitivities of neurones in lateral geniculate nucleus of macaque." *The Journal of Physiology*, vol. 357, no. 1, pp. 219–240, 1984.

[59] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Sequential deep learning for human action recognition," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7065 LNCS, pp. 29–39, 2011.

[60] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[61] N. Ballas, L. Yao, C. Pal, and A. Courville, "Delving deeper into convolutional networks for learning video representations," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–11, 2016.

[62] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734, 2014.

[63] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[64] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[65] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 5999–6009, 2017.

[66] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, no. Mlm, pp. 4171–4186, 2019.

[67] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[68] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[69] R. A. Rensink, "The dynamic representation of scenes," *Visual cognition*, vol. 7, no. 1-3, pp. 17–42, 2000.

[70] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," *Advances in Neural Information Processing Systems*, vol. 3, no. January, pp. 2204–2212, 2014.

[71] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2. IEEE, 2005, pp. 60–65.

[72] D. Alexey, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv: 2010.11929*, 2020.

[73] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[74] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," *Advances in neural information processing systems*, vol. 31, 2018.

[75] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[76] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[77] Y. Hasson, "Kinetics_i3d_pytorch: Inflated I3D network with Inception backbone, weights transferred from TensorFlow," [Online]. Available: https://github.com/hassony2/kinetics_i3d_pytorch, 2017, accessed: 2024-12-10.

[78] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 976–11 986.

[79] S. Jiang, B. Sun, L. Wang, Y. Bai, K. Li, and Y. Fu, "Skeleton aware multi-modal sign language recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3413–3423.

[80] F. A. Research, "Fvcore: A collection of core functionality for PyTorch projects," [Online]. Available: https://github.com/facebookresearch/fvcore, 2019, accessed: 2024-11-17.

[81] V. Bazarevsky, "BlazePose: On-device real-time body pose tracking," *arXiv preprint arXiv:2006.10204*, 2020.

[82] Google AI, "MediaPipe hand landmarker," [Online]. Available: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker, 2024, accessed: 2024-12-23.

[83] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4724–4732.

[84] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[85] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[86] S. Yan, "Spatial Temporal Graph Convolutional Networks (ST-GCN) for Skeleton-Based Action Recognition in PyTorch," [Online]. Available: https://github.com/yysijie/st-gcn, 2018, accessed: 2024-12-23.

[87] J. Zhang, "Decord: An efficient video loader for deep learning with smart shuffling that's super easy to digest," [Online]. Available: https://github.com/dmlc/decord, Jan. 2019, accessed: 2024-09-07.

[88] S. Jiang, B. Sun, L. Wang, Y. Bai, K. Li, and Y. Fu, "Sign language recognition via skeleton-aware multi-model ensemble," *arXiv preprint arXiv:2110.06161*, 2021.

[89] L. N. Smith, "Cyclical learning rates for training neural networks," *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, no. April, pp. 464–472, 2017.

[90] Gugger, "How Do You Find A Good Learning Rate," [Online]. Available: https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html, Mar. 2018, accessed: 2024-09-06.

[91] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[92] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[93] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006. SPIE, 2019, pp. 369–386.

[94] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[95] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[96] S. Narang, G. Diamos, E. Elsen, P. Micikevicius, J. Alben, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–12, 2018.

[97] fast.ai, "Mixed precision training," [Online]. Available: https://docs.fast.ai/callback.fp16.html, accessed: 2022-06-10.

[98] PyTorch, "Distributed Data Parallel," [Online]. Available: https://pytorch.org/docs/master/notes/ddp.html, 2019, accessed: 2024-09-09.

[99] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[100] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

[101] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017.

[102] L. Shi, Y. Zhang, J. Cheng, and H. Lu, "Skeleton-based action recognition with multi-stream adaptive graph convolutional networks," *IEEE Transactions on Image Processing*, vol. 29, pp. 9532–9545, 2020.

[103] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[104] S. Albanie, G. Varol, L. Momeni, T. Afouras, J. S. Chung, N. Fox, and A. Zisserman, "BSL-1K: Scaling up co-articulated sign language recognition using mouthing cues," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*. Springer, 2020, pp. 35–53.

[105] H. Hu, W. Zhao, W. Zhou, Y. Wang, and H. Li, "SignBERT: Pre-training of hand-model-aware representation for sign language recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 11 087–11 096.

[106] Y. Cai, L. Ge, J. Liu, J. Cai, T.-J. Cham, J. Yuan, and N. M. Thalmann, "Exploiting spatial-temporal relationships for 3d pose estimation via graph convolutional networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2272–2281.

[107] S. Javier, E. Meinhardt-llopis, and G. Facciolo, "TV-L1 Optical Flow Estimation," *Image Processing On Line*, vol. 1, no. 1, pp. 137–150, 2013.

[108] S. Chandel and contributors, "Torchsummary: Keras style model.summary() in PyTorch," [Online]. Available: https://github.com/sksq96/pytorch-summary, 2018, accessed: 2024-11-18.

[109] O. Kopuklu, N. Kose, A. Gunduz, and G. Rigoll, "Resource efficient 3d convolutional neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[110] J. Lin, C. Gan, and S. Han, "Tsm: Temporal shift module for efficient video understanding," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7083–7093.

[111] G. Hinton, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[112] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.