LIGHTWEIGHT AMERICAN SIGN LANGUAGE RECOGNITION USING A DEEP LEARNING APPROACH

Yohanes Satria Nugroho, Chuan-Kai Yang and Yuan-Cheng Lai

Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan

ABSTRACT

Sign Language Recognition is a variant of Action Recognition that consists of more detailed features, such as hand shapes and movements. Researchers have been trying to apply computer-based methods to tackle this task throughout the years. However, the methods proposed are constrained by hardware limitations, thus limiting them from being applied in real-life situations.

In this research, we explore the possibilities of creating a lightweight Sign Language Recognition model so that it can be applied in real-life situations. We explore two different approaches. First, we extract keypoints and use a simple LSTM model to do the recognition and get 75% of Top-1 Validation Accuracy. We used the lightweight MoViNet A0 model for the second one and achieved 71% of Top-1 Test accuracy. Although these models achieved worse results than the state-of-the-art I3D, their complexity in terms of FLOPs is far better.

KEYWORDS

Sign Language Recognition, Lightweight Model, Keypoints Estimation

1. INTRODUCTION

Due to their disability to hear or speak, often time deaf people will face barriers in communicating with others. Sign Language is a language used mainly by deaf people to communicate with others. Instead of using spoken words, Sign Language combines hand gestures, facial expressions, and body posture [1] as features to represent a word. In 2011, The American Community Survey estimated that around 11 million of Americans, or 3.6% of the population, were deaf or hard of hearing [2]. This number alone proves that the barriers are getting wider and must be solved.

As science and computer technology advance, many researchers have been trying to tackle the Sign Language Recognition problem using advanced technology. Most of these researchers tried to use methods such as Kinect [3] and gloves [4, 5], or adopt computer science techniques, such as machine learning [6, 7] and action recognition deep learning [8] fine-tuned on a sign language dataset. Although these methods might promise high-accuracy recognitions, they have limitations preventing them from being used for everyday life use-cases. For example, people will find it very inconvenient to bring Kinect or motion gloves attached to a computer outside of the laboratory so that they can recognize the sign language signed by deaf people. Thus, having a

David C. Wyld et al. (Eds): NLPML, AIAP, SIGL, CRIS, COSIT, DMA -2023
pp. 75-90, 2023. CS & IT - CSCP 2023 DOI: 10.5121/csit.2023.130807

small device that a person can bring anywhere and anytime, such as a smartphone, to do the recognition would be more desired.

On the other hand, most action recognition models are computationally expensive due to their complexity in return for high-accuracy prediction. While most Desktop PC might be equipped with powerful hardware, smartphones come with a limited hardware capacity. Even recently released high-end smartphones are still inferior in terms of computation power compared to mid-end desktop PCs. This limitation makes it difficult to run a model locally on smartphones. Due to the models' complexity, researchers considered reducing the complexity of the model [9] by decreasing the number of layers used in the model. This approach is doable, however, it comes at the cost of reducing the accuracy significantly. One option that might be considered is to use a client-server approach [10]. Generally speaking, a user will capture the input stream using his or her smartphone (client) and send it to the server. The input stream will then be fed to the heavy-weight model to produce the prediction result and send back the result to the client. However, this approach still has a downside since an internet connection is still needed so that the client and server can communicate with each other.

Motivated by the limitations of current research, this work aims to provide a lightweight sign language recognition model that can be run locally on a smartphone with reasonable recognition accuracy. To address the limitations mentioned in the previous section, this work will explore two different approaches as our main contributions. First, we try to explore the possibilities of combining rich-feature input, that is, keypoints, with a simple stack of Long-Short Term Memory (LSTM) layers associated with low computation complexity. Secondly, we use a pre-trained lightweight action recognition model and fine-tune the model using a sign language dataset.

For this research, we use the publicly available MS-ASL Dataset [11], an American Sign Language Dataset published by Microsoft back in 2019. To limit the scope of this research, we use the subset of this dataset, namely MS-ASL100, which only contains the 100 most frequent classes. In the first approach, this work uses MediaPipe [12] Holistic keypoint Detection as the feature extractor and trains a baseline model from scratch. In the second approach, we explore the ability of a lightweight action recognition model to perform the sign language recognition task, namely Movinet-A0 [13], pre-trained on the Kinetics-600 dataset, and fine-tune the model using the MS-ASL100 dataset.

The rest of this paper is organized as follows. Section two, related works from previously done research will be presented to provide readers with brief information needed about Sign Language Recognition, Human Pose Estimation, Action Recognition, and MoViNet Architectures. Section three, information about the proposed system, such as architectures and methods used, will be explained. Section four, experimental setups, experiment results, and limitations, will be explained. In the last section, we will summarize the result of this research and plan future works so that this research can be improved to a higher level.

2. RELATED WORKS

2.1. Action Recognition

Many researchers have been proposing architectures that result in good classification accuracy result. The most common one is to use the 3D convolution architecture. Unlike the 2D convolution (Figure 1 b) on multiple channels or frames that collapse all temporal information due to its kernel depth that matches the total length (i.e., channels × frames) of the data, 3D convolution (Figure 1 c) uses a kernel less than the data length and adds a depth movement to the

kernel to process other input frames. This approach enables the 3D convolutions to retain the temporal information, unlike the 2D convolutions. Examples of this implementation are C3D [14] and I3D [15]. However, as mentioned in [16], performing 3D convolutions will cost expensive computation.

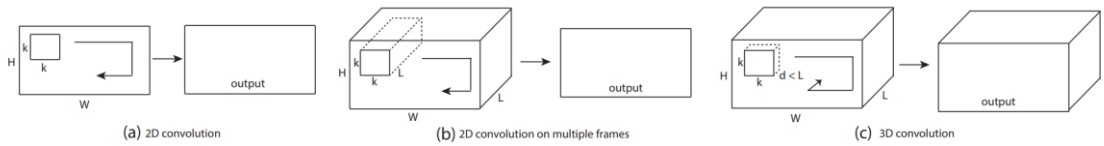

Figure 1. Comparison between 2D convolution and 3D convolution operation [14]

Different from the 3D convolutions that learn the temporal features directly from the stack of RGB frames as the input, Two-Stream architecture [17] uses the optical flow. Instead of using a single convolution block (stream), this model uses two separate convolution streams. The model learns the temporal features by calculating the optical flow between a pair of two consecutive RGB frames. These optical flows are then passed to a dedicated convolutional block to learn the temporal features. The results from both spatial and temporal convolutional blocks are fused to get the final result. However, as mentioned in the original research paper, calculating optical flow is computationally expensive, and thus performing the calculation on-the-fly will cause a bottleneck. An example of a Two-Stream architecture can be seen in Figure 2.

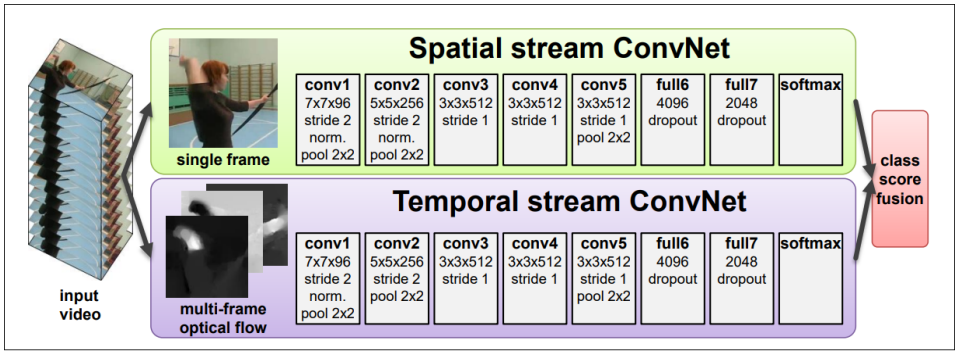

Figure 2. Visualization of two-stream architecture [17]

Another approach is 2+1D convolution, as proposed in [18]. Unlike 3D convolution, which learns the spatial and temporal features at the same block, 2+1D convolution decomposes the learning process into two separate operations, a 2D spatial convolution and a 1D temporal convolution. This decomposition gives advantages such as providing more nonlinearities compared to the normal 3D convolution and makes the model capable of providing more complex functions. A comparison of 3D convolution with 2+1D convolution can be seen in Figure 3.

2.2. Sign Language Recognition

Many researchers have been trying to adopt action recognition architectures to tackle the sign language recognition task. The most common action recognition adopted for this task is the 3D Convolution Neural Network (3D CNN) used in [19] and [20]. These works extracted the RGD-D data from the input videos and fed them to the 3D CNN layers for the feature extraction process. The features extracted by the 3D CNN layers are then passed to a fully connected layer for the classification process.

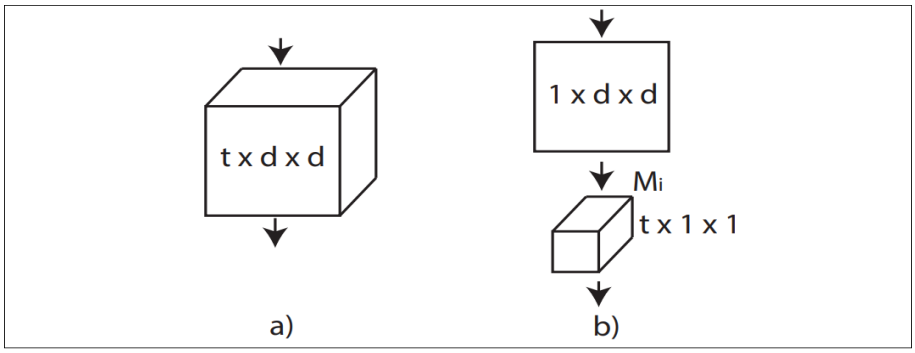

Figure 3. Visualization of two-stream architecture [18]. a) A full 3D convolution operation. b) 2+1D convolution operation

Another example is by using a Video Transformer architecture as used in [21]. This architecture takes 16 frames of RGB data from the input video and encodes them using the 2D CNN encoder. The encoded information is then passed to the decoder block, which is built by utilizing a combination of multi-head attention and convolutional block. The outputs of the decoder are then passed to a fully connected layer to perform classification.

A hybrid approach is also used by [22]. This approach was made by combining a 2D CNN block, namely VGG-S and GoogLeNet, as the spatial feature extractor and a stack of LSTM layers as the temporal feature extractor. The outputs of the LSTM layers were then passed to a fully connected layer to calculate the classification result.

2.3. Human Pose Estimation

Many research areas in the computer vision domain have emerged as science progresses. Human Pose Estimation is one of them. Human Pose Estimation is a task that focuses on detecting the human body parts given the input image [23]. Examples of currently existing human pose estimation are OpenPose [23], MediaPipe [12], and YOLO-Pose [24], which utilizes YOLO architecture as the pose detector.

As mentioned in [23], the most common approach to do the human pose estimation is by detecting the human from the input image using a human detector. The human detected by the human detector will be passed to the pose detector to estimate their body parts. The output of these pose detectors is usually in the form of the x, y, and z coordinates for each of the body part detected. Not only limited to the relatively big body parts, such as legs, arms, and neck, pose detectors also can be trained to estimate keypoints of the smaller body parts, such as joints in fingers or face area.

2.4. MoViNets

Although deep learning in the video recognition research field has been advancing, formulating a lightweight model that can be run on smaller devices, such as smartphones and edge devices, has yet achieved a reasonable accuracy, thus still being challenging.

Mobile Video Networks (MoViNets) [13] were created with a goal of creating a model with an excellent trade-off in terms of complexity and accuracy. MoViNets used TuNAS [25], a Tunable Network Architecture Search (NAS) framework, to find the most efficient model architecture. This process resulted in MoViNet-A2 and performed the search space scaling resulting in other

variants of MoViNets, namely the A0, A1, A3, A4, A5, and A6, where A0 is their lightest model, and A6 is their heaviest model.

Although MoViNets still uses 3D convolution operations, their complexity, in terms of FLOPs per video, is comparable to the recent lightweight models. MoViNet A0 has been proven to have FLOPs as low as the MobileNetV3-Small while having its Top-1 video accuracy slightly better than the Mobile- NetV3-Large. Compared to the X3D models, the Top-1 accuracy of MoViNet-A0 is slightly lower than the X3D-XS variant while having its FLOPs per video significantly lower.

3. PROPOSED METHOD

3.1. Models

As mentioned in the first section of this paper, the main contribution of this research is to provide a model with complexity light enough to be run on a smartphone with reasonable accuracy. This subsection will explain the models we are going to use

3.1.1. Keypoints + LSTM

Research such as in [15, 22], uses a CNN-based architecture, such as VGG-S model, to extract the spatial feature from each input frame, and pass them into the LSTM layer to model the temporal feature. In [15], the CNN+LSTM model was trained on 3 different action recognition datasets. The CNN+LSTM model from [15] achieved 81.0% accuracy using the UCF-101 dataset, a standard benchmark dataset for the action recognition task. The similar architecture from [22], failed to demonstrate good performance when used on the ASL100 split of the MS-ASL dataset and achieved only 13.33% of average per class accuracy as mentioned in [11]. Looking at the result from the research we mentioned previously, we believe that combining CNN and LSTM might be a good and straightforward solution on a simple action recognition dataset, such as UCF-101. We also believe that this combination failed to achieve a good result on the ASL100 dataset because the spatial feature extractor could not extract important features to be passed to the LSTM layer.

Based on this hypothesis, we tried to change the LSTM layer's input. We remove the CNN spatial feature extractor with MediaPipe, a human pose estimator from Google. The MediaPipe will extract the keypoints of the signers from each frame and pass them as the input to the LSTM model. The LSTM model we used consisted of 3 LSTM layers with 64, 512, and 512 hidden units. We added Batch Normalization and ReLU activation functions in each LSTM layer. The output of the final LSTM layer is passed to Dense Layers to perform classification using softmax activation function. The architecture of the LSTM model we use is shown in Figure 4.

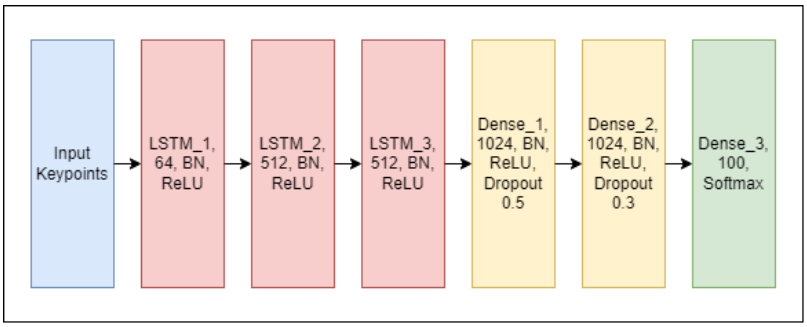

Figure 4. LSTM block architecture.

3.1.2. MoViNets

Another option we want to explore is to use a mobile device-friendly model and fine-tune the model on the MS-ASL dataset. We decided to use the MoViNet model since this model showed an outstanding result in its publication paper. Since this model is already light enough, we decided to follow their implementation using the RGB input instead of using keypoints as the input as we did on the LSTM model.

In this research, we chose the lightest variant of MoViNet, namely the A0 variant, among the other variants. Aside from it being the lightest variant, the A0 variant offers a comparable Top-1 accuracy compared to heavier variants, such as the A1 variant, while having one-third of the complexity in terms of FLOPs. We use the official implementation of MoViNet-A0 from TensorFlow Hub. We unfreeze the parameters in the last 3 layers so that the parameter in these layers can be fine-tuned to learn the feature from the dataset we used. This approach can be seen in Figure 5.

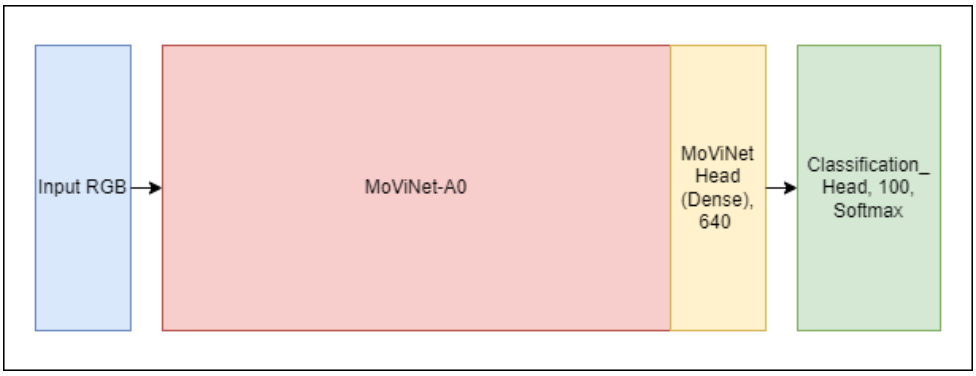

Figure 5. MoViNets approach.

3.2. Dataset

The dataset used for this research is the MS-ASL dataset, the first large-scale publicly available American Sign Language dataset published by Microsoft in 2019. The original dataset consists of 1000 classes of words, signed by over 200 signers, and more than 7000 distinct videos, split into more than 25000 annotated data.

Unlike other datasets that have all the video files included when we download them, MS-ASL will only give us JSON files. These files contain information about the dataset, such as class, signing start time, signing end time, resolution, and source of the video so that we can download

these videos manually. As a result, manually downloading all the videos from their source was inevitable. We downloaded the videos on June 2021. At the time of downloading, most videos were removed or restricted, thus resulting in a loss of dataset. The download process resulted in 4735 videos for 1000 classes. Considering the number of available videos compared to the overall classes, we took the 100 most frequent classes and created a subset from the original dataset with 870 distinct videos and 3956 data.

Since the length of a single piece of data is most likely different from other data, we decided to split 1 single piece of data into several parts consisting of T frames. We follow the implementation from [15] and used T equals 16 frames for this case. We then split a single data into several splits of 16 consecutive frame-long data that were randomly taken [11], resulting in 24967 16 frame-long data.

3.3. Dataset Processing

As the data in the dataset are in the form of raw video data, it is necessary to preprocess them before the training process. In this research, we will explore two different approaches: Keypoints as input to a simple LSTM model and RGB as input to a lightweight MoViNet model. Since these two models have different input typse, we also processed the raw video using two different approaches.

3.3.1. Keypoints Input

For the LSTM model, the x, y, and z coordinates of the predicted keypoints were extracted from each video frame using the MediaPipe Pose Estimation library, specifically the Holistic Pose Estimation. This pose estimation returns a total of 1629 keypoints, where each keypoint is composed of a normalized value of x, y, and z coordinates. We set the minimum detection confidence of Holistic Pose Estimation to a value of 0.75, minimum tracking confidence to 0.75, and used the model with the model complexity set to 0. We horizontally flip the video frame and estimate the keypoints for both the original and flipped frame to acquire more data samples.

Having (1629 × 3) keypoints per frame as input to the model will increase the complexity of the model. Although other research, such as [21], mentioned that facial features such as facial expression, mouth shape, and eyebrow shape are important features in sign language, we believe that hand shape and the sequence of the movements are more important. Thus, we retained only the important key points to ensure that the model still has the ideal complexity. We took all 42 hand keypoints for both hands, two sets of 3 pose keypoints, consisting of the shoulder, the elbow, and the wrist keypoint for both arms, and removed all 468 face mesh keypoints. The keypoints extracted can be seen in Figure 6.

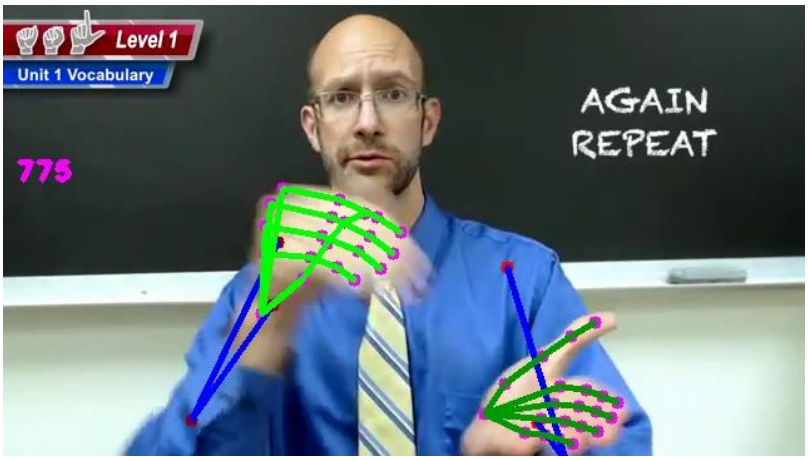

Figure 6. Keypoints extracted by MediaPipe.

We also added two angle values for both arms as additional features as the input to the model. We believe that adding this feature will enrich the input feature and help the model to learn better. We then calculate the angle using Equation 1. The gradient between the shoulder and the elbow keypoints is denoted as m_1. The gradient between the wrist and the elbow keypoints is denoted as m_2. These aforementioned features will be concatenated, resulting in 146 features for each frame. We then take 16 randomly sampled frames from the original and the flipped video. Finally, we will have data with a shape of (2 x 16 x 146)

$$\theta = \arctan((m_2 - m_1) / (1 + m_1 m_2)) \quad \text{(Equation 1)}$$

3.3.2. RGB Input

For the MoViNet model, we extract RGB value from the raw video using the Python programming language and OpenCV library. As suggested in [13] and to limit the model's complexity, we resize the width and height of extracted RGB into (172 × 172) pixels for each frame. Horizontal flipping was applied to the resized RGB to ensure that each data comes with both right-handed and left-handed examples. We then randomly took 16 frames from each right-handed and left-handed data, resulting in data with the shape of (2 × 16 × 172 × 172 × 3).

4. EXPERIMENTS & RESULTS

4.1. Training

Due to hardware limitations, performing experiments on the models using our resources would be challenging. To make the experiments easier, we used the Google Colaboratory service as our main platform. The specifications we used are specified in Table 1.

In this research, we will use the Top-K accuracy as the metrics in every experiment. The Top-K accuracy metric counts how many times the ground truth ranked among the top K labels predicted by the model. For our experiments, we specifically choose the K value to be 1 and 5, since the Top-1 and Top-5 accuracy are the most common metrics used in the Action Recognition field.

Table 1. Hardware Specifications.

Property	Specification
OS	Linux
CPU	Intel® Xeon® Processor 2.20 GHz
GPU	Nvidia A100-SXM4 40 GB
RAM	89.6GB

4.1.1. LSTM

For the LSTM model, we had to train the model from scratch. The model was trained for 80 epochs using Adam optimizer with 0.001 as the initial learning rate value. We use the exponential learning rate decay to decrease the initial learning rate in every epoch. We split the whole keypoints dataset into 80% for being the training set and 20% for being the validation set in the training phase. We use a batch size of 512 in every training step. Finally, we use the categorical cross-entropy as the loss function of the LSTM model. The comparison between Top-1 and Top-5 can be seen in Figure 7a.

From both the Top-1 and the Top-5 comparisons, we can see that the validation results from both metrics made no meaningful fluctuations starting from epoch 55. The LSTM model achieved 75% in terms of Top-1 validation accuracy and 89% in terms of Top-5 validation accuracy. We then perform another experiment to see whether the dataset size affects the model's accuracy. Unlike the first experiment, we remove 20% random data from the whole keypoints dataset. We then split the remaining dataset into 80% for training and 20% for validation. We used the same training parameters for this experiment as the previous one. This experiment resulted in 69% of Top-1 validation accuracy and 86% of Top-5 validation accuracy. The result of this experiment can be seen in Figure 7b.

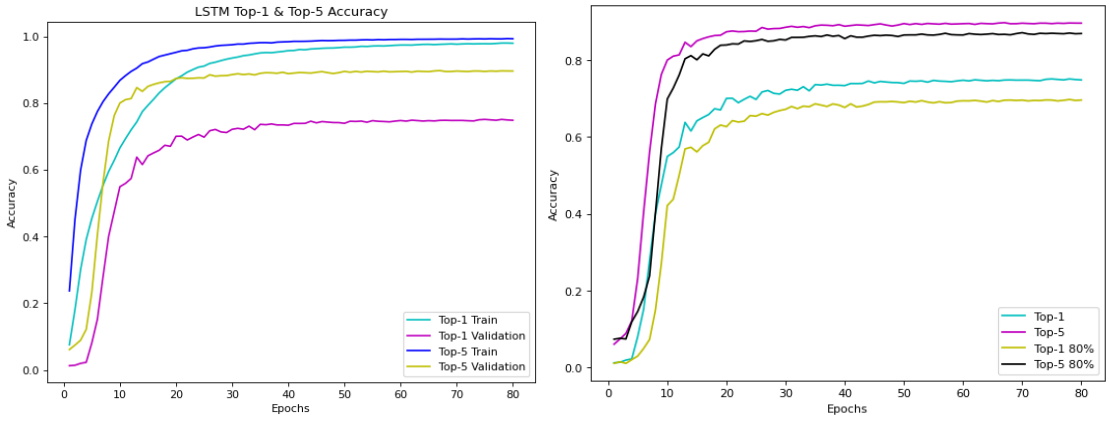

Figure 7. a – left) LSTM Model Top-1 & Top-5 Accuracy Comparison; b – right) Effects of reducing data size in accuracy

We can see from Figure 7b that there were accuracy differences of 5\% in terms of Top-1 accuracy and 3\% in terms of Top-5 accuracy between the first and the second experiments. Based on these results, we can conclude that the dataset's size affects the model's validation accuracy. This means that, with the number of data we currently have, the result in Figure 7a is the best currently we can achieve, and increasing the number of data used for training might improve the model's accuracy even further.

4.1.2. MoViNet

For the MoViNet model, we used the MoViNet implementation from Tensorflow Hub. We used the MoViNet A0 model, which has been pretrained with the Kinetics-600 Action Recognition Dataset. We then fine-tune the A0 with the RGB dataset. For the fine-tuning process, we let the weights and biases parameters in the last 3 layers to be trainable. One advantages of using a pretrained model is that usually a good result can be reached on a small number of epochs. Therefore, unlike the LSTM where we use 80 epochs to train the model from scratch, it is possible to train the MoViNet on 15 epochs due to the model is already pretrained on a similar task.

As the LSTM model, we used the categorical cross-entropy as this model's loss function. We used the RMSProp with an initial learning rate of 0.01 and Cosine Decay to reduce the learning rate in every epoch. We also use a batch size of 8 per training step. Unlike its keypoints counterpart, the RGB dataset we use is more massive, with a total size of (49934 × 16 × 172 × 172 × 3) compared to (49934 × 16 × 146). To avoid running into the out-of-memory problem while training, we equally split the whole RGB dataset into 10 parts. Each split still consists of 100 classes, but each class have only one-tenth of data from the original dataset. We then tried to train the A0 model iteratively using 3 different splits. We split each of these 3 dataset split into 80% for training and 20% for validation.

From our experiments, training the model further using other parts of the dataset did not significantly improve accuracy, and often led to worse performance. We suspect that the model overwrote the optimal weights and biases with a worse set of parameters because of the learning rate used. However, finding an optimal learning rate for the later experiment was harder than the earlier one. We achieved 72% in terms of Top-1 validation accuracy and 86% in terms of Top-5 validation accuracy after 3 training iterations, where each iteration means a different dataset split was used to train the model. The training results is shown in Figure 8.

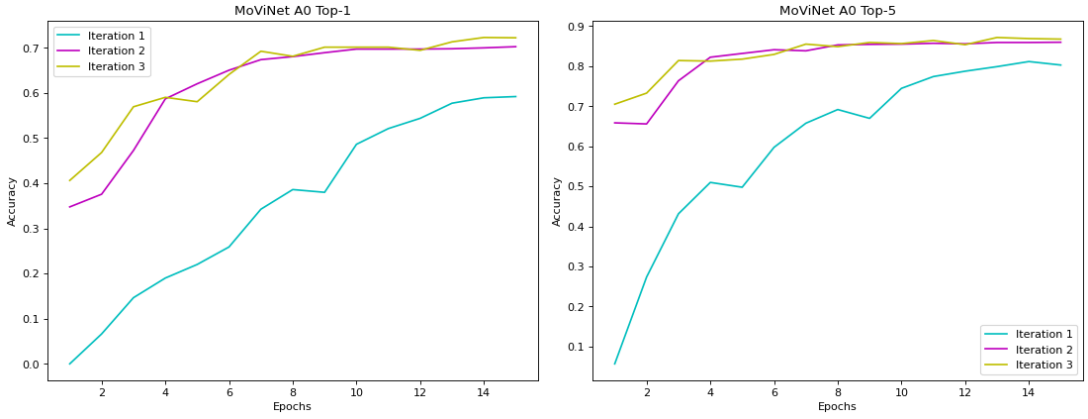

Figure 8. a – left) MoViNet A0 Top-1 Validation Accuracy Comparison; b – right) MoViNet A0 Top-5 Validation Accuracy Comparison

We also tried to fine-tune the MoViNet A3 to see whether the heavier variant of MoViNet would achieve better accuracy. We performed training for MoViNet A3 using the same setting as the MoViNet A0. At the end of the third training iteration, the MoViNet A3 achieved 74% of Top-1 validation accuracy and 88% of Top-5 validation accuracy. Based on this result, we chose to use the A0 variant instead of the A3 since the difference in terms of accuracy is not significant, yet

the FLOPs increased drastically [13]. The comparison between the MoViNet A0 and A3 can be seen in Figure 9.

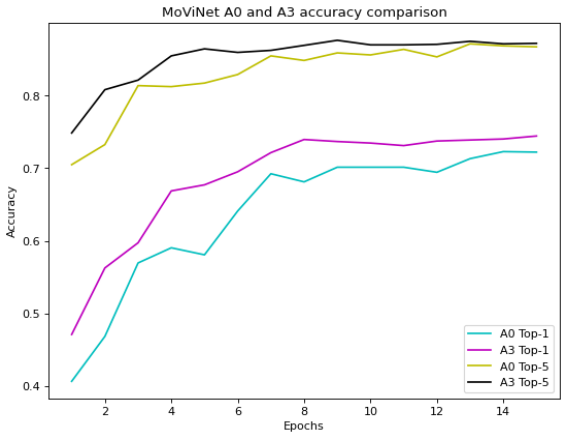

Figure 9. MoViNet A0 and A3 Validation Accuracy Comparison

4.2. Experimental Result

Since we used only 3 out of 10 parts of the RGB dataset to fine-tune the A0, we still have unused 7 parts. We combine these RGB dataset splits and perform a test phase on our MoViNet A0 model. From this experiment, the model achieved 71% of average Top-1 accuracy and 87% of average Top-5 accuracy. These results are not far from the validation result in the training phase with a 1% margin of error. We create a histogram based on the Top-1 accuracy to see how many classes achieved low result. The histogram can be seen in Figure 10.

Based on Figure 10, we can see that 70 classes achieved Top-1 accuracy above 70%. Although this result is relatively satisfactory, there were some classes with scores lower than the others. We take "man" and "woman" classes with average Top-1 accuracy of 38% and 42%, for example. At first, we thought that the number of training data for these classes were not sufficient by having around 200 data for each class. Therefore, the model could not classify them correctly. It turned out that "coffee" and "computer" have a similar number of training data, yet they achieved far better Top-1 and Top-5 test accuracy.

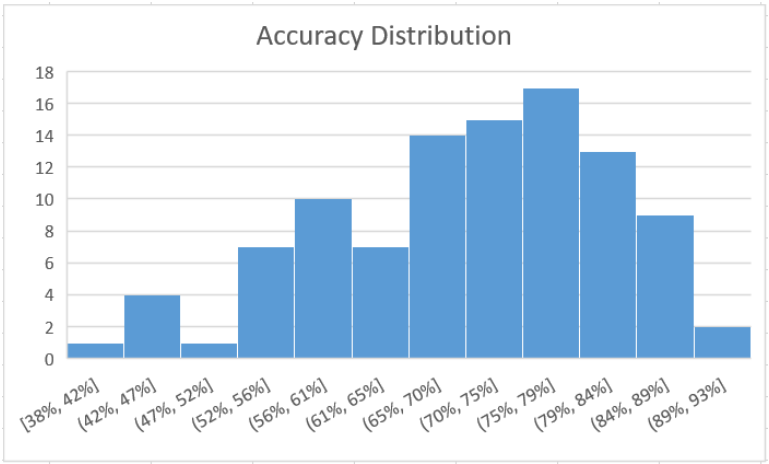

Figure 10. MoViNet A0 Accuracy Histogram

We tried to investigate the Top-1 and Top-5 results of the "man" class further. These results can be seen in Table 2. Having these results, we proceed to check videos where a signer signs "Man", "Father", "Boy", "Fine", "Like", and "Woman" manually. It turned out that these categories have some similarities compared to the "Man" categories. The similarities we found are regarding the hand shape and the movement. Based on these findings, we think that our fine-tuned MoViNet A0 is still not discriminative enough in this case due to the facing of many similar features. We also found similar results when we investigated the "Woman" sign, thus making this category also achieved relatively low results. Feature similarities of aforementioned categories can be seen in Figure 11.

Table 2. Classes recognized as "Man". #Top-1 and #Top-5 indicate how many times a class considered as Top-1 and Top-5 respectively.

Rank	**Class**	**# Top-1**
1	Man	73
2	Father	17
3	Boy	9
4	Fine	8
5	Like	8

Rank	**Class**	**# Top-5**
1	Man	125
2	Woman	61
3	Father	57
4	Boy	47
5	Fine	29.

For the LSTM model, we could not perform a test like what we did for the MoViNet model since we already take the whole keypoints dataset for the training purpose. We could not reserve some parts of this dataset due to decreasing the number of data used in the training phase would affect the model's accuracy, just as what we proved in the LSTM model's second experiment. However, based on the MoViNet's slight difference between validation and test results, we believe the LSTM model will also achieve test results relatively close to its validation results.

We also compared other models from [11] with our LSTM and MoViNet A0 models. We note that the results in [11] were from models trained on the ASL-100 split, which is the same as the dataset we used. Table 3 shows the comparison in terms of average class Top-1 and Top-5 accuracy.

Figure 11. Similarities between ”Man”, ”Father”, ”Boy”, ”Fine”, ”Like”, and ”Woman”

Table 3. Classes recognized as "Man". #Top-1 and #Top-5 indicate how many times a class considered as Top-1 and Top-5 respectively.

Method	**Top-1**	**Top-5**
Naïve Classifier	1%	5%
VGG + LSTM ([22], [26])	14%	34%
HCN [27]	46%	74%
Re-Sign [28]	45%	-
I3D [15]	81%	95%
Keypoints + LSTM (ours)	75%	89%
MoViNet-A0 (ours)	71%	87%

Although both of our models achieved lower in both Top-1 and Top-5 accuracy compared to the I3D model, our models have significantly lower complexity in terms of FLOPs, where the LSTM

requires 108 MFLOPs, and the MoViNet requires 2710 MFLOPs, while the I3D requires 108 GFLOPs. Our models even have the same complexity compared to the MobileNetV3-Small + TSM used in [13]. Since MobileNetV3 is already famous for its capability to run on a low-power device, we believe that our model can also run on low-power devices as well. The FLOPs comparison between these models is shown in Table 4. We also note that the complexity of the Keypoints + LSTM approach is still unknown due to the unknown complexity of the MediaPipe to get the keypoints which is denoted by *C*. However, we believe the total complexity of the Keypoints + LSTM approach is still relatively close to the MoViNet A0. Thus, we choose the Keypoints + LSTM as our best approaches since the inference time is faster than the MoViNet A0.

Table 4. Complexity comparison

Method	**Complexity (MFLOPs)**	**Inference (ms)**
I3D [15]	107890	-
Keypoints + LSTM (ours)	*C* + 108	28.14
MoViNet-A0 (ours)	2710	172.4

4.3. Limitations

As mentioned previously, the major problem we face is the number of data we have is limited due to many reasons. And as we proved in the previous experiment, having less data for training will lower the model's ability to predict the sign language performed by a user. It also means that if we have more data for training, there is a possibility for the model to learn better and increase its ability.

Another problem is that the MediaPipe oftentimes could not perform pose estimation correctly, or even could not read the pose at all. Some of the videos already have challenges, such as occlusion, bad lighting, and low resolution, which could worsen MediaPipe's performance. In fact, MediaPipe can also fail in some videos with fewer visual challenges.

5. CONCLUSION & FUTURE WORKS

5.1. Conclusion

In this research, we tried to explore the possibilities of creating a lightweight model for American Sign Language using the MS-ASL American Sign Language Dataset. We finally came up with two different approaches. The first one is to use pose estimation to get the keypoints in each input video frame and pass it into the LSTM model. The second approach is to extract the RGB value from each input video frame and pass it to the MoViNet-A0, a lightweight video action recognition model.

For the LSTM model, we achieved 75% in terms of Top-1 validation accuracy and 89% in terms of Top-5 validation accuracy. For the MoViNet-A0 model, we achieved 71% of average class Top-1 accuracy and 87% of average class Top-5 accuracy. Although our models achieved lower accuracy than other state-of-the-art models, such as I3D, which had achieved these results under significantly lower complexity. Our LSTM model only requires 108 MFLOPs, and 2710 MFLOPs for the MoViNet-A0, compared to 108 GFLOPs on the I3D. Our models also offer a lower complexity in terms of FLOPs when compared to MobileNet-V3, which is already known as a mobile device-friendly model with around 2800 MFLOPs. These results prove that both

models we proposed theoretically could be run on a mobile device locally while having a reasonable ability to predict American sign language signed by the users.

5.2. Future Works

The main objective of this work is to find out the possibility of creating lightweight models that could be run on mobile devices locally. Although this research has answered this question, this answer is however still a theory. Thus, further implementing these models into real mobile device applications is still required. Although the accuracy of the models we proposed is relatively reasonable, we believe there are ways to improve the accuracy even further. We believe that having more data with better quality or having more advanced yet still lightweight models might improve the result in the lightweight sign language recognition research area.

Combining these sign language recognition models with the Natural Language Processing field would further improve their usability. Instead of just recognizing word by word, the NLP model will process the input words and output a proper sentence that a normal person can easily understand.

REFERENCES

[1] O. M. Sincan and H. Y. Keles, "AUTSL: A Large Scale Multi-modal Turkish Sign Language Dataset and Baseline Methods," CoRR, vol. abs/2008.00932, 2020.

[2] CCDHHDB, "DHHDB Demographics," 2022. [Online]. Available: https://ccdhhdb.com/wp-content/uploads/2022/09/DHHDB-Demographics.pdf. [Accessed December 2022].

[3] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton and P. PrestI, "American Sign Language Recognition with the Kinect," in 13th International Conference on Multimodal Interfaces, ICMI, Alicante, Spain, 2011.

[4] S. A. Mehdi and Y. N. Khan, "Sign language recognition using sensor gloves," in 9th International Conference on Neural Information Processing, ICONIP, Singapore, 2002.

[5] 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, article 63, 2009.

[6] T. Starner and A. Pentland, "Visual recognition of american sign language using hidden markov models," 1995.

[7] T. Starner and A. Pentland, "Real-time American Sign Language recognition from video using hidden Markov models," in International Symposium on Computer Vision - ISCV, Coral Gables, FL, USA, 1995.

[8] P. Adarsh, P. Rathi and M. Kumar, "YOLO v3-Tiny: Object Detection and Recognition using one stage improved model," 6th International Conference on Advanced Computing and Communication Systems (ICACCS), 2020.

[9] A. A. Hosain, P. S. Santhalingam, P. Pathak, H. Rangwala and J. Košecká, "Hand Pose Guided 3D Pooling for Word-level Sign Language Recognition," IEEE Winter Conference on Applications of Computer Vision (WACV), 2021.

[10] R. A. Abdul Rahman and C.-K. Yang, Mobile application for real-time bird sound recognition using convolutional neural network, 2021.

[11] H. R. V. Joze and O. Koller, "MS-ASL: A Large-Scale Data Set and Benchmark for Understanding American Sign Language," British Machine Vision Conference, 2019.

[12] MediaPipe, "MediaPipe," [Online]. Available: https://mediapipe.dev/. [Accessed 12 11 2022].

[13] D. Kondratyuk, L. Yuan, Y. Li, L. Zhang, M. Tan, M. Brown and B. Gong, "MoViNets: Mobile Video Networks for Efficient Video Recognition," CVPR, 2021.

[14] D. Tran, L. Bourdev, R. Fergus, L. Torresani and M. Paluri, "C3D: Generic Features for Video Analysis," ArXiv, vol. abs/1412.0767, 2014.

[15] J. Carreira and A. Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017 .

[16] S. Xie, C. Sun, J. Huang, Z. Tu and K. Murphy, "Rethinking Spatiotemporal Feature Learning For Video Understanding," arXiv:1712.04851, 2017.

[17] K. Simonyan and A. Zisserman, “Two-Stream Convolutional Networks for Action Recognition in Videos,” arXiv:1406.2199, 2014.
[18] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun and M. Paluri, “A Closer Look at Spatiotemporal Convolutions for Action Recognition,” IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018 .
[19] L. Jing, E. Vahdani, M. Huenerfauth and Y. Tian, “Recognizing American Sign Language Manual Signs from RGB-D Videos,” arXiv:1906.02851, 2019.
[20] L. Pigou, S. Dieleman, P.-J. Kinderma and B. Schrauwen, “Sign Language Recognition Using Convolutional Neural Networks,” European Conference on Computer Vision ECCV, pp. 572-578, 2014.
[21] M. De Coster, M. Van Herreweghe and J. Dambre, “Sign Language Recognition with Transformer Networks,” in 12th Language Resources and Evaluation Conference, Marseille, France, 2020.
[22] R. Cui, H. Liu and C. Zhang, “A Deep Neural Framework for Continuous Sign Language Recognition by Iterative Training,” IEEE Transactions on Multimedia, vol. 21, no. 7, 2019.
[23] Z. Cao, G. Hidalgo, T. Simon, S. Wei and Y. Sheikh, “OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 43, pp. 172-186, 2021.
[24] D. Maji, S. Nagori, M. Mathew and D. Poddar, “YOLO-Pose: Enhancing YOLO for Multi Person Pose Estimation Using Object Keypoint Similarity Loss,” 2022.
[25] G. Bender, H. Liu, B. Chen, G. Chu, S. Cheng, P. Kindermans and Q. Le, “Can weight sharing outperform random architecture search? An investigation with TuNAS,” arXiv:2008.06120, 2020.
[26] R. Cui, H. Liu and C. Zhang, “Recurrent Convolutional Neural Networks for Continuous Sign Language Recognition by Staged Optimization,” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1610-1618, 2017.
[27] W. Zhu, C. Lan, J. Xing, W. Zeng, Y. Li, L. Shen and X. Xie, “Co-occurrence Feature Learning for Skeleton based Action Recognition using Regularized Deep LSTM Networks,” The 30th AAAI Conference on Artificial Intelligence (AAAI-16), 2016.
[28] O. Koller, S. Zargaran and H. Ney, “Re-Sign: Re-Aligned End-to-End Sequence Modelling with Deep Recurrent CNN-HMMs,” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

AUTHORS

Yohanes Satria Nugroho received a Master of Information Management degree from the National Taiwan University of Science and Technology, Taiwan, in 2023. His research interests include deep learning, computer vision, and mobile application.

Chuan-Kai Yang received the B.S. and M.S. degree in mathematics and computer science from National Taiwan University, Taipei, Taiwan, in 1991 and 1993, respectively, and the Ph.D. degree in computer science from Stony Brook University, Stony Brook, NY, USA, in 2002. He is a Professor with the Department of Information Management, National Taiwan University of Science and Technology, Taipei. His research interests include computer graphics, visualization, multimedia systems, and computational geometry.

Yuan-Cheng Lai received his Ph.D. degree in Computer Science from National Chiao Tung University in 1997. He joined the faculty of the department of Information Management at National Taiwan University of Science and Technology in 2001 and has been a professor since 2008. His research interests include wireless networks, network performance evaluation, network security, and artificial intelligence.