Electronic Engineering and Informatics
G. Izat Rashed (Ed.)

doi:10.3233/ATDE240085

Isolated Sign Language Recognition Based on Deep Learning

Junjie WANG[1]
School of Mechanical, Shandong University, Weihai, 264200, China

Abstract. Communicating with hearing-impaired individuals poses a significant challenge. However, with the advancement of computer vision, automatic sign language recognition (SLR) is gradually addressing this issue and has made significant improvements. One of the key challenges in SLR lies in accurately capturing and interpreting the subtle nuances and variations in sign language gestures. In this study, our focus is on recognizing isolated sign language using the LSA64 dataset, which is a small-scale dataset of Argentinian isolated sign language. We concatenated CNN and LSTM into an end-to-end sign language recognition model for an isolated sign language recognition dataset, recognition of the Argentine Sign Language dataset (LSA64). We achieved promising results in our study, obtaining a high accuracy rate of nearly 97% while ensuring that the model remained compact in size.

Keywords. Sign Language Recognition, LSA64, Convolutional Neural Networks, Recurrent Neural Networks

1. Introduction

Communication is essential in people's life. Good communication facilitates better understanding of each other's intentions during interpersonal interactions, and this holds true for individuals with hearing loss, speech impairments, and deafness as well, if not more so. According to estimates, around 430 million people have moderate to profound hearing loss, predominantly affecting their better ears. Based on surveys and estimates by the World Health Organization, the prevalence of hearing loss varies across different regions worldwide, with a significant majority of these individuals residing in low- and middle-income countries. Sign language, like spoken language, serves as a means of communication between individuals and can assist in facilitating communication and interaction among individuals with speech impairments and hearing difficulties. It can also help bridge the communication gap between them and individuals with normal hearing. However, the majority of individuals with normal hearing are not familiar with sign language and may have had little to no exposure to it.

Additionally, learning sign language can be challenging and requires dedicated effort and energy. Therefore, individuals who are deaf and mute still face significant challenges in communicating and interacting with the majority of people who have normal hearing, especially in some scenarios that require faster processing, such as hospitals and banks.

Although people with hearing impairments can write and type on mobile phones to communicate with others, that is still too inefficient. With the rapid progress of deep

[1] Junjie WANG, School of Mechanical, Shandong University, e-mail: 202000800559@mail.sdu.edu.cn

learning techniques and the availability of large-scale sign language datasets, SLR has gradually been making progress in addressing this issue. In isolated sign language recognition, each gesture represents a single word or phrase, while in continuous sign language recognition, a sequence of consecutive gestures represents a complete sentence. As shown in Figure 1, SLR is receiving more and more attention from researchers in recent years.

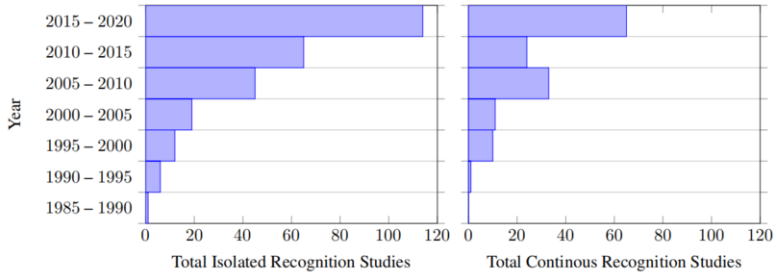

Figure 1. The number of SLR results published between 1983 and 2020. [1]

Based on the input modality, sign language recognition systems can be broadly classified into two categories: vision-based and data-glove-based systems. The vision-based system can collect real-time three-dimensional motion information and temporal changes in gestures. It can apply recognition algorithms to process the data, offering fast recognition speed and high accuracy, but the device is complex, expensive, and restrictive for the operator, inconvenient to wear, so vision-based sign language recognition has become mainstream. Garcia and Viesca [2] used the CNN method to translate American Sign Language (ASL) into the alphabet. Cate et al. [3] used RNN to conduct time sequence modeling for video features of sign language and realized the recognition of isolated words in 95 classes of sign language. In the preceding years, CNN models and RNN models have been widely used for SLR.

We used CNN and RNN (recurrent neural networks) to recognize sign language. For the temporal and spatial features of sign language dataset videos, we employed two different models for training. CNN was trained on the frames obtained from the video sequences of train data to extract spatial features. Then, the CNN extracts feature maps from the frames, and these feature maps are fed into an RNN to capture the temporal information among the frames in the sequence. We concatenated CNN and LSTM into an end-to-end SLR model for an isolated SLR dataset, recognition of the Argentine Sign Language dataset (LSA64).

We are committed to solving SLR for fast SLR scenarios, such as banks, hospitals and other scenarios. We explore different network structures to obtain more effective SLR feature representations. We explore a lightweight neural network to reduce the difficulty of terminal device deployment while ensuring accuracy requirements as much as possible.

2. Related Work

2.1. Sign language recognition methods

Many papers focusing on sign language recognition have been published in various

journals and conferences, aiming to assist individuals who rely on sign language for communication. Adewuyi et al. [4] conducted a study in which they integrated electromyography (EMG) data from finger and arm muscles to classify different types of hand grip and finger movements. Molchanov et al. pioneered the application of 3D-CNN for dynamic gesture recognition, introducing this method at CVPR 2015. They used multi-scale data as input to the network and constructed a two-way subnetwork to extract the spatiotemporal feature set of gestures. The classification results of the two subnetworks were then scored and fused, achieving good recognition results in the context of automatic driving. In 2017, Li et al. [5] introduced a novel hand-shape descriptor and utilized LSTM-based temporal modeling with these descriptors, leading to precise recognition outcomes in the domain of Chinese sign language recognition. Kopuklu et al. [6] presented a data-level fusion approach at CVPR2018, which combines motion information with static images and feeds the resulting spatiotemporal features into a CNN network for classification. The results of this approach were highly satisfactory. Lin et al. [7] proposed a novel approach combining a masked RES-C3D network and an LSTM network, which achieved a recognition accuracy of 68.42% on the ChaLearn dataset. Devineau et al. proposed a CNN architecture based on hand skeleton data for recognizing 3D dynamic gestures. They utilized parallel convolutions to process the positional sequences of hand skeletal joints and extracted hand connectivity nodes, resulting in a high-accuracy outcome.

2.2. Deep learning models

Our model utilized Convolutional neural networks (CNNs) and Long Short-Term Memory (LSTM) architectures, and we conducted comparative experiments using separate CNN and 3D-CNN models.

CNNs excel at capturing local spatial patterns in data and are particularly well-suited for image classification tasks. These networks are designed to process input data in the form of images and are able to identify patterns in the data that can be used to classify them. One advantage of CNNs is that they are relatively insensitive to image rotations or translations, thanks in part to the use of pooling layers. This means that a rotated or translated version of an image will often be classified in the same way as the original image. Because CNNs are highly effective at extracting spatial information from images, our model compared the performance of pre-trained deep learning models including VGG16, ResNet50 [8], ResNet101[9], and InceptionV3 from the Keras library with a CNN model we developed from scratch.

Recurrent neural networks (RNNs) are capable of utilizing the information in the sequence itself, making them suitable for recognition tasks. Unlike other neural networks, RNNs have loops and their output is dependent on the combination of current input and previous output. However, a known drawback of RNNs is their inability to learn long-term dependencies in practice. To overcome this limitation, our model employed LSTM units, a variation of RNN. LSTM units possess strong abilities to learn across time intervals, as research has shown that they can effectively handle sequences spanning over 1000 steps and maintain good performance even on noisy or non-compressible input sequences.

3DCNNs use a 3D kernel to perform 3D convolutions on the input data. The kernel moves across the input data in 3 dimensions, allowing the network to capture the spatial and temporal variations present in the data. They can be used for tasks such as action recognition in videos, medical diagnosis and treatment, and industrial defect detection in

3D imaging data. Compared to traditional methods for analyzing 3D data, 3DCNNs can learn complex spatiotemporal features automatically from the input data without the need for handcrafted features.

2.3. Traditional video classification methods

Given that our dataset comprises video data, our task can be construed as that of video classification. Video classification is a highly challenging task because video sequences encompass two types of sequences: the spatial information carried by the frame images within the video, and the temporal information among the frames extracted from the video. Traditional deep learning methods for video classification typically involve combining 2D CNNs with temporal pooling or RNNs such as LSTMs or GRUs. The 2D CNNs are used to extract spatial features from individual frames of a video, while the temporal pooling or RNNs are used to model the temporal evolution of these features over time. Another common approach is to use 3D CNNs, which can directly model both spatial and temporal features of a video. These methods have been widely used in video classification tasks, such as action recognition and SLR.

3. Our Approach

3.1. Our models

In our main experiments, we experimented with seven different models we name CNN1+LSTM, CNN2+LSTM, CNN8+LSTM, VGG16+LSTM, ResNet50+LSTM, ResNet101+LSTM and InceptionV3+LSTM, chosen for diversity of approaches. All the models mentioned in the previous studies utilize LSTM for training the model on temporal features. However, the primary differences among these models lie in the network architecture of the convolutional layers, which are responsible for processing spatial information in the videos. We utilized a range of convolutional layer models to extract spatial information from the videos. These models encompassed single-layer convolution, two-layer convolution, eight-layer convolution, and well-known architectures such as inceptionv3, VGG16, resnet50, and resnet101. Figures 2, 3, and 4 depict the architectural diagrams of all our models.

3.2. Data preprocessing

The LSA64 dataset comprises a total of 3200 videos, where each video is created by 10 regular participants performing 64 distinct hand gestures with 5 repetitions each. In each class of videos, we used 5 videos from the first 8 participants as the training set, and the remaining videos were used as the test set. Each video comprises approximately 120 to 170 frames. Subsequently, we performed equidistant frame sampling on both the training and test sets, followed by resizing each image to a size of (150, 80, 3). In our experiment, we evaluated the impact of extracting 8 frames and 24 frames at equidistant intervals. Given the use of different models, CNN, CNN+LSTM, and 3DCNN, it was necessary to employ distinct input data for each model type. For the CNN model, we used each individual frame image extracted from the videos as a separate input. However, for the CNN+LSTM and 3DCNN models, we combined all the extracted images from a single

video into a sequential sequence and used it as a single input. Finally, to ensure that the labels of the data match the model's output, we convert them into one-hot encoding.

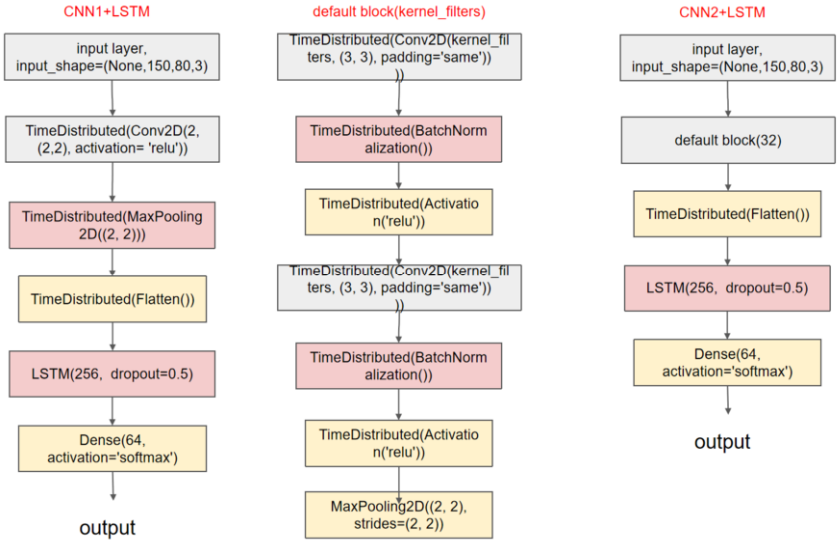

Figure 2. The CNN1+LSTM, CNN2+LSTM models and default block

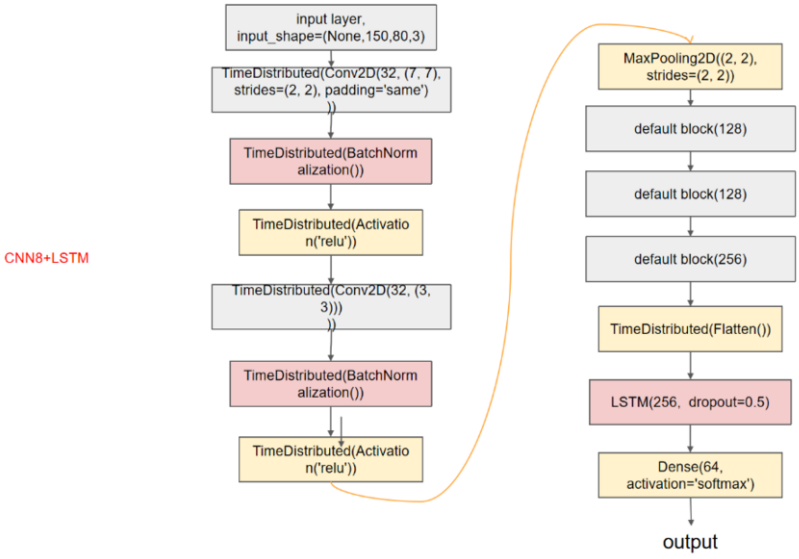

Figure 3. The CNN8+LSTM model

3.3. Methods

To ensure that these models can take a sequence of frames extracted from a single video as input, we employed the TimeDistributed layer in Keras. TimeDistributed is a layer wrapper in Keras that allows applying a layer to every temporal slice of an input. Every

input should be at least 3D, and the dimension of index one of the first input will be considered the temporal dimension. This wrapper is useful when processing sequences of vectors or sequences of sequences. In our model, we first pass the image sequence through a CNN to extract features. Then, we apply the Flatten operation to flatten the features and convert them into a one-dimensional vector. Next, we input this one-dimensional vector into an LSTM to process the sequential temporal information. Finally, we use a fully connected layer with a SoftMax activation function for classification. This layer takes the output from the LSTM layer and maps it to the corresponding class probabilities.

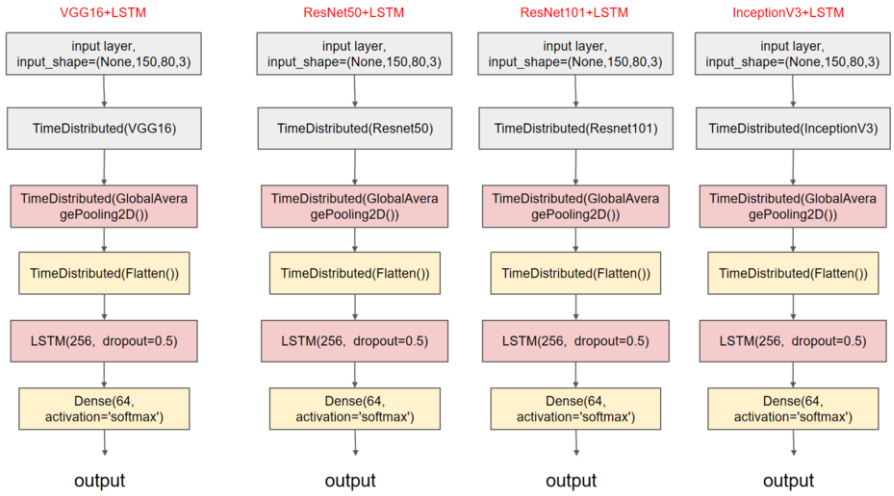

Figure 4. Architecture diagrams for the last four models

The SoftMax activation function ensures that the predicted probabilities sum up to 1, enabling us to interpret the output as the predicted probabilities for each class. This allows us to classify the input sequence into the appropriate class based on the highest probability.

3.4. Loss function

We utilize Categorical cross-entropy as the loss function for our models. Categorical cross-entropy is a commonly used loss function in deep learning, particularly for multi-class classification tasks. It measures the dissimilarity between the predicted probability distribution and the true labels of the classes.

In the context of multi-class classification, the categorical cross-entropy loss function compares the predicted class probabilities outputted by the model with the one-hot encoded true labels. It quantifies the discrepancy between the predicted probabilities and the actual labels by computing the cross-entropy, which is a measure of information entropy.

The categorical cross-entropy loss function encourages the model to minimize the difference between the predicted probabilities and the true labels, thereby maximizing the model's ability to correctly classify instances into their respective classes. It provides a gradient signal that guides the model's parameter updates during the training process, enabling it to learn meaningful representations and make accurate predictions.

By optimizing the categorical cross-entropy loss function, the model learns to assign higher probabilities to the correct class labels and lower probabilities to incorrect ones. This loss function is widely used in neural network architectures for multi-class classification tasks and has proven to be effective in training models to achieve high accuracy in classifying diverse and complex datasets.

4. Experiments

4.1. Dataset

We evaluate our method on the LSA64 dataset. This sign language dataset was created by researchers to aid in the study of SLA for Argentine Sign Language (LSA). The LSA64 dataset comprises a total of 3200 videos, where each video is created by 10 regular participants performing 64 distinct hand gestures with 5 repetitions each. These 64 hand gestures are all commonly used in LSA. Subjects were recorded while wearing gloves of different colors, with each hand assigned a distinct color. Among the 64 categories of sign language gestures, 42 gestures are performed using a single hand, while the remaining 22 gestures involve both hands as shown in Figures 5 and 6. In the first 42 gestures, the participants wore a red glove on their right hand only, while in the remaining gestures, the participants wore green gloves on their left hands. This means that in the sign language videos where only one hand is used, our model needs to ignore the information from the other hand when recognizing the gestures.

Figure 5. Here are sample frames extracted from the LSA64 videos. The background is clean, and the individuals are clearly visible. The images exhibit minimal noise.

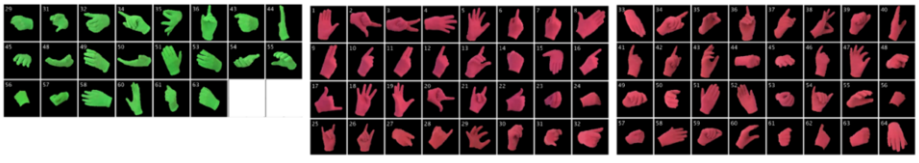

Figure 6. The gesture samples in LSA64 consist of hand gestures with the left hand wearing a green glove (on the left) and hand gestures with the right hand wearing a red glove (in the middle and on the right).

4.2. Model evaluation metrics

Multiple metrics are applicable to machine learning and SLR to gauge model performance. Accuracy, precision, recall and f-score are the four most common metrics. For the proposed system, accuracy is the chosen metric for comparing the experiments. Accuracy is the ratio of the number of correct predictions to the total number of input samples as given by Equation 1 [10]. In addition, to explore the lightweight nature of the model, we recorded the average time taken per epoch for each experiment, which served as an additional evaluation metric. Furthermore, for all models, we conducted experiments using the same hardware setup, which consisted of an RTX

A5000 24G*1 GPU and an AMD EPYC 7371 CPU. This ensured consistent computational resources for fair comparison and reliable performance evaluation across the models.

$$Accuracy = \frac{number\ of\ correct\ classifications}{total\ number\ of\ classifications\ attempted} \tag{1}$$

4.3. Main experiments result

All the experiments in this paper were conducted using Keras under the TensorFlow 2.5 version. In all experiments, we consistently utilized the pre-defined training set as the training set and the test set as the validation set. The primary aim of the training process was to enhance the accuracy of the validation set, thereby ensuring the model's ability to generalize and enabling the evaluation of its performance. For all models in the main experiments, we employed the RMSprop optimizer with a learning rate of 0.001. We utilized categorical cross-entropy as the loss function and trained the models for 300 epochs. For models that take an input of an 8-frame sequence f, we set the batch size to 32. For models that take an input of a 24-frame sequence, we made a modification by setting the batch size to 8. Furthermore, we explored various techniques to improve the model's performance, including using pre-trained weights from ImageNet, freezing the backbone network, adjusting the learning rate and batch size, and experimenting with different loss functions and optimizers. Finally, we present the best results achieved through these optimizations. The main experiment result is shown in Table 1.

Table 1. Main experiment result

Models	Frames	Accuracy	Training time/epoch
CNN1+LSTM	8	73.59%	6 s
CNN2+LSTM	8	81.563%	10 s
CNN8+LSTM	8	96.72%	9 s
VGG16+LSTM	8	77%	14 s
ResNet50+LSTM	8	93.59%	31 s
ResNet101+LSTM	8	93.28%	50 s
InceptionV3+LSTM	8	94.80%	21 s
CNN1+LSTM	24	75.63%	14 s
CNN2+LSTM	24	82.227%	21 s
CNN8+LSTM	24	93.906%	22 s
VGG16+LSTM	24	81%	31 s
ResNet50+LSTM	24	88.44%	94 s
ResNet101+LSTM	24	88.75%	144 s
InceptionV3+LSTM	24	93.75%	65 s

4.4. Comparative experiments result

We conducted comparative experiments using CNN and 3DCNN models. We named these models as follows: CNN2, VGG16, ResNet50, ResNet101, and InceptionV3. In comparison to the main experiment, we also employed the RMSprop optimizer with a learning rate of 0.001, utilized categorical cross-entropy as the loss function and trained the models for 300 epochs while training the CNN model. Given that CNN models are not designed to handle sequential image data directly for classification, we adopted a strategy where each extracted frame was treated as an individual input. The pre-processed training set, initially prepared for the CNN model, served as our training set, while the test set was used as the validation set. Our main objective throughout the experiments was to enhance the model's performance on the validation set. Specifically, for the dataset where videos were divided into 8 frames, we set the batch size to 256. For the dataset where videos were divided into 24 frames, we set the batch size to 192. This ensures that the amount of data entering the neural network in each batch is the same as that of the models used in the main experiment. We decomposed a video into frames and fed them individually into the model. The model produced predictions for each frame, and we determined the final output for the video by selecting the result with the highest frequency among the model's output predictions. We employed this approach to evaluate the CNN models.

We utilized a modified version of 3DCNN based on C3D architecture. However, due to the issue with the size of convolutional kernels, we only utilized the dataset with 24 frames for this particular experiment. Additionally, we used the Adam optimizer and set the learning rate to 1×10^{-5}, and the loss function and epochs remained the same as in the main experiment. The comparative experiments result is shown in Table 2.

Table 2. Comparative experiment result

Models	Frames	Accuracy	Average training time /epoch
CNN2	8	32.97%	-
VGG16	8	57.03%	-
ResNet50	8	65.00%	-
ResNet101	8	65.00%	-
InceptionV3	8	80.63%	-
CNN2	24	20.09%	-
VGG16	24	48.28%	-
ResNet50	24	66.09%	-
ResNet101	24	59.84%	-
InceptionV3	24	77.81%	-
3dcnn	24	87.16%	30s

5. Conclusion

Over all the proposed system performed well. The CNN8+LSTM model performed exceptionally well, achieving an accuracy of 96.72% on the dataset with 8 frames. It also demonstrated good results in terms of training time per epoch (representing the lightweight nature of the model). This is the most outstanding model in our experiment. However, the other models in the main experiment also performed exceptionally well. Moreover, compared to training with CNN alone, there was a significant improvement

in performance. This highlights the remarkable ability of LSTM to handle the temporal features between images. The training speed and performance of 3DCNN were also impressive.

As observed from the experimental results mentioned above, increasing the number of frames in the dataset from 8 to 24 did not lead to a significant improvement in the model's accuracy. In fact, there was a slight decrease in performance. Based on our analysis, the LSA64 dataset exhibits minimal variation in actions, and a smaller number of frames can effectively capture the motions in each video.

Our main conclusion is that a lightweight and accurate model can help address the challenges presented by the LSA64 dataset.

In future work, we will further explore improving the accuracy of the model while maintaining its lightweight nature. We will also investigate more complex isolated sign language datasets. Once we establish a more accurate and convenient model for isolated sign language datasets, we will extend our research to address challenges in continuous sign language datasets.

References

[1] Oscar Koller. (2020) Quantitative survey of the state of the art in sign language recognition. arXiv preprint, 09918.
[2] Brandon Garcia, Sigberto Alarcon Viesca. (2016) Realtime American sign language recognition with convolutional neural networks. In: Convolutional Neural Networks for Visual Recognition, 2:225-232.
[3] Hardie Cate, Fahim Dalvi, Zeshan Hussain. (2017) Sign language recognition using temporal classification. arXiv preprint, 1701.01875.
[4] Adenike A Adewuyi, Levi J Hargrove, Todd A Kuiken. (2015) An analysis of intrinsic and extrinsic hand muscle EMG for improved pattern recognition control. In: Transactions on Neural Systems and Rehabilitation Engineering, 24(4):485-494.
[5] Xiaoxu Li et al. (2017) Chinese sign language recognition based on shs descriptor and encoder-decoder lstm model. In: 12th Chinese Conference Biometric Recognition, Shenzhen, 719-728.
[6] Okan Kopuklu, Neslihan Kose, Gerhard Rigoll. (2018) Motion fused frames: Data level fusion strategy for hand gesture recognition. In: Conference on computer vision and pattern recognition workshops, 2103-2111.
[7] Chi Lin et al. (2018) Large-scale isolated gesture recognition using a refined fused model based on masked res-c3d network and skeleton lstm. In: International conference on automatic face & gesture recognition, 52-58.
[8] Kaiming He et al. (2016) Deep residual learning for image recognition. In: Conference on computer vision and pattern recognition, 770-778.
[9] Christian Szegedy et al. (2016) Rethinking the inception architecture for computer vision. In: Conference on computer vision and pattern recognition, 2818-2826.
[10] Mohammad Hossin, Md Nasir Sulaiman. (2015) A review on evaluation metrics for data classification evaluations. In: International journal of data mining & knowledge management process, 5(2):1.