A Modern Approach for Sign Language Interpretation Using Convolutional Neural Network

Chapter · August 2019

DOI: 10.1007/978-3-030-29894-4_35

6 authors, including:

Pias Paul
North South University
2 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Nabeel Mohammed
North South University
41 PUBLICATIONS **214** CITATIONS
SEE PROFILE

Sifat Momen
North South University
37 PUBLICATIONS **146** CITATIONS
SEE PROFILE

A Modern Approach for Sign Language Interpretation Using Convolutional Neural Network

Pias Paul(✉), Moh. Anwar-Ul-Azim Bhuiya, Md. Ayat Ullah, Molla Nazmus Saqib, Nabeel Mohammed, and Sifat Momen

Department of Electrical and Computer Engineering, North South University, Plot-15, Block-B, Bashundhara, 1229 Dhaka, Bangladesh
{paul.pias,anwar.bhuiyan,ayat.ullah,nazmus.saqib,nabeel.mohammed, sifat.momen}@northsouth.edu

Abstract. There are nearly 70 million deaf people in the world. A significant portion of them and their families use sign language as a medium for communicating with each other. As automation is being gradually introduced to many parts of everyday life, the ability for machines to understand the act on sign language will be critical to creating an inclusive society. This paper presents multiple convolutional neural network based approaches, suitable for fast classification of hand sign characters. We propose two custom convolutional neural network (CNN) based architectures which are able to generalize 24 static American Sign Language (ASL) signs using only convolutional and fully connected layers. We compare these networks with transfer learning based approaches, where multiple pre-trained models were utilized. Our models have remarkably outperformed all the preceding models by accomplishing 86.52% and 85.88% accuracy on RGB images of the ASL Finger Spelling dataset.

Keywords: Image processing · CNN · Transfer learning · ASLR · Finger Spelling dataset

1 Introduction

A language that needs manual communication and involvement of body language to convey meaning as opposed to conveyed sound patterns is known as sign language. This can involve a simultaneous combination of handshapes, orientation, and movement of the hands, arms or body, and different facial expressions to fluidly express a speaker's thoughts. In some cases, sign language is the only method that is used to communicate with a person with hearing impairment. Sign languages such as the American Sign Language (ASL), British Sign Language (BSL), Quebec Sign Language (LSQ), Spanish sign language (SSL) differ in the way an expression is made. They share many similarities with spoken languages, which is why linguists consider sign languages to be a part of natural languages.

A. C. Nayak and A. Sharma (Eds.): PRICAI 2019, LNAI 11672, pp. 431–444, 2019.
https://doi.org/10.1007/978-3-030-29894-4_35

Sign Language Recognition (SLR) system which is required to perceive gesture-based communications, has been widely studied for years. It provides a way to help deaf/mute individuals interact easily with technology. However, just like speech recognition, this is not an easy task. However, recent advances in computer vision, particularly the use of convolutional neural networks (CNN), has created opportunities to create effective solutions to problems previously thought to be almost unattainable. In this paper, we present multiple CNN-based models to classify 24 characters from the ASL Finger Spelling Dataset. We present models which were custom made for this problem, as well as models which leverage transfer learning. One of our custom models achieved a test accuracy of 86.52%, which is better than the current best published result.

2 Related Works

In 2013 Pugeault and Bowden [18] proposed an interactive keyboard-less graphical user interface that can detect hand shapes in real time. In that work, they used a Microsoft Kinect device for collecting both appearance and depth images, OpenNI+NITE framework for hand detection and tracking, features based on Gabor filters and a Random Forest classifier for classification. From the ASL dataset, which was also proposed in that paper, they have ignored and discarded images of letter j and z since both of these letters require motion and used leftover 48000 images; 50% for training and 50% for validation. Using both appearance and depth images together brought them better classification result compared to the usage of appearance and depth information separately.

Tripathi et al. have proposed a continuous hand gesture recognition system [29]. In their approach, keyframes were extracted using gradient-based methods and HoG features were used for actual feature extraction. For classification, several distance metrics were used including City Block, Mahalanobis, Chess Board, Cosine, etc. They created a dataset using 10 sentences signaled by 5 different people. They found that using a higher number of bins for HoG resulted in better performance and the best performance was found when Euclidean distance employed.

Masood et al. [15] proposed a method to bridge the gap for the people who do not know and want to communicate using sign languages through isolated sign language recognition using methods based on computer vision. They used an Argentinean dataset (LSA) with 2300 video samples and substantial gesture variation with 46 categories. Their model used the Inception-v3 pre-trained CNN, and combined with the use of Long Short Term Memory (LSTM) for sequence predictions. They tried 3 models such as a single layer of 256 LSTM units, a wider Recurrent neural network(RNN) network with 512 LSTM units, a deep RNN network consisting of 3 layers with each 64 LSTM units. Empirically, they found the model with 256 LSTM units gave the best performance. Two approaches were taken for training, one was a prediction approach in which predictions of frames made by CNN were fed as input to the LSTM. In the other

approach, the output of the pooling layers was directly fed into the LSTM. The second approach gave a better result with an accuracy of 95.2%.

3 Experimental Setup

This section provides details of the setup used for the experiments performed. We initially present the dataset on which we will train and compare the different models. This is followed by a brief description of the data preprocessing and partitioning. The proposed models are discussed next, which includes descriptions of the custom models as well as the transfer learning techniques.

3.1 Dataset

The work is based on ASL Finger Spelling dataset that consists of images which were obtained from 5 different users. In the proposed dataset [18], images were obtained in 2 different ways, each user was asked to perform 24 ASL static signs which were captured in both color and depth format. There are a total of **131,670** number of images where **65,774** images have RGB channels and rest are depth images that contain the intensity values in the image which represent the distance of the object or simply depth from a viewpoint. The reason behind choosing American Sign Language (ASL) for this work was that ASL is widely learned as a second language and the dataset contains sign from only using one hand which reduces the task of over-complicated feature extraction. Here, the dataset comprises 24 static signs which have similar lighting and background excluding the letters j and z since these 2 letters require dictionary lookup and involve motion (Table 1).

Table 1. Types of images collected from each user

User	Image type	
	RGB	Depth+RGB
A	12,547	25,118
B	13,898	27,820
C	13,393	26,810
D	13,154	26,332
E	12,782	25,590

3.2 Data Preprocessing and Feature Extraction

From the total of **5** user samples, **4** were considered in such a way that the proposed dataset [18] was divided into two parts. First part is Dataset-A which contains only color images and the other one is Dataset-B which contains both

Table 2. Preparing the dataset

Image type	Training set	Validation set	Label
RGB	26,547	26,445	**DataSet-A**
Depth+RGB	53,142	52,938	**DataSet-B**

depth and color images. This is shown in Table 2. In both the DataSet-A and DataSet-B, images from users C and D were used as the training set and images from user A and B were used to make validation/test set. As the images were of different sizes, all of them were re-sized to 200×200 pixels. Pixel color values were re-scaled between 0 and 1 and then each image was normalized by subtracting the mean (Fig. 1).

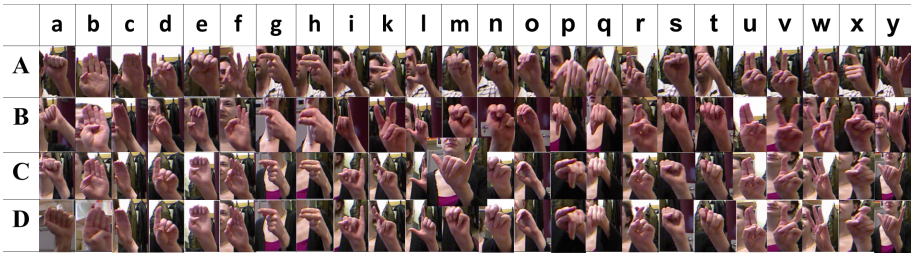

Fig. 1. Illustration on the variety of the dataset where each column represents images of individual letters that has been collected from 4 different users.

To increase the amount of training data, each training image was augmented using the transformations mentioned in Table 3. The augmentations were applied single (not compositionally) and were only applied to RGB images. The validation data were not augmented per say, but were modified.

Table 3. Augmentation techniques applied on Dataset-A and Dataset-B

Training data set		Validation data set	
Arguments	Parameters	Arguments	Parameters
Rescale	1./255	Rescale	1./255
Center-Cropped	True	Center-Cropped	True
Shear_Range	0.2 Degree		
Zoom_range	0.1		
Random_Rotation	20 Degree		
Horizontal_Flip	True		
Height_Shift_Range	0.1		
Width_Shift_Range	0.1		
Fill_Mode	Nearest		

3.3 The Proposed Architecture

Table 4 shows the details of the two custom models used for comparison. Both models were trained and tested on DataSet-A and DataSet-B. For Custom-Model-A, conv3-32 means respective field size 3 and number of channels 64. The images were resized to 128×128 dimension which will go through the convolutional layers. The model uses LeakyReLU as activation function. We found LeakyReLU to work better than RelU for this model after experimentation. Apart from using max pooling, Global Average Pooling (GAP) was also used to downsample the input dimension from each layer using a 2×2 window. After flattening the output of the last pooling layer was passed through four fully connected layers with the final layer having 24 neurons for the 24 classes. The last layer also uses the softmax activation function.

In custom-Model-B, 2×2 strided convolution [27] was used to reduce the size of the output feature maps instead of the more commonly used pooling techniques. Surprisingly, for this model, our tests showed better performance using the RelU activation function (an investigation looking into the discrepancy is currently under progress and will be reported in a later paper). Batch normalization was also used in this model. This model also flattens the output of the last convolutional layer and forwards the output to four fully connected layers, although the configurations are slightly different from Custom-Model-A.

3.4 Transfer Learning Using Pre-trained Models

Apart from our custom models, we have also experimented using Transfer Learning which leverages the weights or filters of a pre-trained model on a new problem as in the case of most real-world problems when there are insufficient data points to train complex models. The premise is if knowledge from an already trained machine learning model is applied to a different but related problem, it may facilitate the learning process as the model is already trained to identify some potentially useful features.

Figure 2 shows the overall strategy used for transfer learning. This method is one which has been used in many different tasks, where the softmax layer of the original pre-trained model is discarded and replaced by a new classification layer with random weights. All layers except this new one are frozen and then the newly crafted model is trained until the random weights change to be compatible with the rest of the model. Then the frozen layers are unfrozen and the entire model is trained.

For this work, we experimented with five different models all pre-trained on the ImageNet dataset. These are MobileNetV2, NASNetMobile, DenseNet21, VGG16 and VGG19.

3.5 Training Details

We arrived at a set of hyper parameters which worked well through experimentation. Table 5 summarizes this information.

Table 4. Configuration of the customized models

Configuration of the proposed architectures	
Custom-Model-A	**Custom-Model-B**
input (128 × 128**image)**	**input (**200 × 200 **image)**
conv3-32. LeakyReLU conv3-32. LeakyReLU conv3-32. LeakyReLU	conv3-64 Sequential (conv3-64.ReLU, BatchNorm conv3-64.Relu, BatchNorm)
MaxPool (stride=2)	**conv3-64 (stride = 2)**
Dropout(0.6) conv3-32. LeakyReLU conv3-32. LeakyReLU conv3-32. LeakyReLU	Sequential (conv3-64.ReLU, BatchNorm conv3-64.Relu, BatchNorm) Sequential (conv3-64.ReLU, BatchNorm)
MaxPool (stride=2)	**conv3-64 (stride = 2)**
DepthwiseConv3. LeakyReLU BatchNormalization conv3-32. LeakyReLU BatchNormalization	Sequential (conv3-64.ReLU, BatchNorm, conv3-64.Relu, BatchNorm) Sequential (conv3-64.ReLU, BatchNorm, conv3-64.Relu, BatchNorm) Dropout(0.6)
conv3-64. LeakyReLU conv3-64. LeakyReLU conv3-64. LeakyReLU conv3-64. LeakyReLU	**conv3-64 (stride = 2)** Sequential (conv3-64.ReLU, BatchNorm, conv3-64.Relu, BatchNorm) Sequential (conv3-64.ReLU, BatchNorm, conv3-64.Relu, BatchNorm)
MaxPool (stride=2)	**conv3-64 (stride = 2)** **conv3-64 (stride = 2)**
conv3-64. LeakyReLU conv3-64. LeakyReLU conv3-64. LeakyReLU conv3-64. LeakyReLU	Sequential (conv3-64.ReLU, BatchNorm, conv3-64.Relu, BatchNorm) Sequential (conv3-64.ReLU, BatchNorm, conv3-64.Relu, BatchNorm)
GlobalAveragePooling **MaxPool (stride=2)**	**conv3-64 (stride = 2)**
FC-1024 Dropout(0.4) ReLU	Dropout(0.6) FC-576 ReLU
FC-576 Dropout(0.4) ReLU	Dropout(0.6) FC-256 ReLU
FC-256 Dropout(0.4) ReLU	Dropout(0.6) FC-128 ReLU
FC-128 Dropout(0.4) ReLU	Dropout(0.6) FC-64 ReLU
FC-24, softmax	FC-24, softmax

The loss function of choice was categorical cross entropy as shown in Eq. 1, which measures the classification error as a cross entropy loss when multiple categories are in use. Here, the double sum is over the observations i, whose number is N, and the categories c, whose number is C and the term $1_{y_i \in C_c}$

Table 5. Training details

Training details	
Batch Size	64
Input size	200*200*3
Learning Rate	0.001
Optimizer	Adam
Loss Function	Categorical Crossentropy
Epoch	25

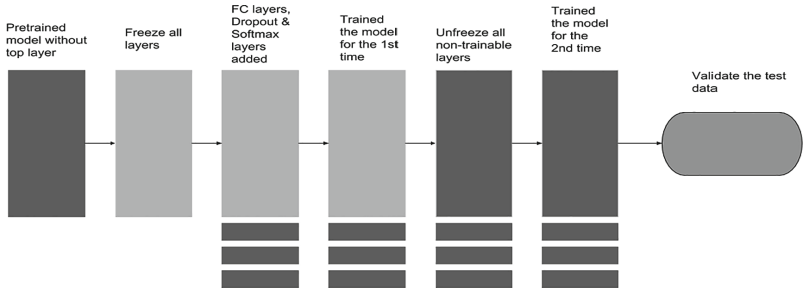

Fig. 2. The proposed transfer learning process

is the indicator function of the ith observation belonging to the cth category. Finally, the probability predicted by the model for the ith observation belongs to which of the cth category is determined by $P_{model}[y_i \in C_c]$.

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} 1_{y_i \epsilon C_c} log P_{model}[y_i \epsilon C_c] \tag{1}$$

For this work the base **learning rate was set to 0.001** with which the network starts to train itself but as mentioned earlier the learning rate is being adapted step wise here by using Eq. 2. Here ***last_epoch***, the value of ***Step_Wise_LR*** will be updated as if an epoch completes all its steps.

$$Step\_Wise\_LR = (base\_lr * gamma * (\frac{last\_epoch}{step\_size})) \tag{2}$$

4 Results

The task of finding a model which will detect the signs based on ASL was divided into two parts. In the first segment, two custom models were built from which accuracy of **86.52%** and in the second segment, an accuracy of **85.88%** was achieved using pre-trained models on Dataset-A.

4.1 Results from Custom Model

To evaluate the model several approaches were taken. At first, two custom models (custom-model-A) and (custom-model-B) were created using the corresponding configurations mentioned in Sect. 3.3. Using the custom-model-A mentioned in Table 6, **77.19% accuracy was achieved** while validating images from Dataset-A. Here to minimize overfitting **40% and 60% of dropout, L2 regularization and Global Average Pooling (GAP)** were used. After **25 epochs** a training accuracy of 96.33% and a validation accuracy of **77.19%** were achieved using 5,214,840 trainable parameters for RGB images.

Table 6. Results obtained using the custom models

Model name	No. of trainable parameters	DataSet-A		DataSet-B	
		Training accuracy (%)	Validation accuracy (%)	Training accuracy (%)	Validation accuracy (%)
Custom-Model-A	5,214,840	96.33	77.19	86.54	**66.79**
Custom-Model-B	**428,728**	**98.54**	**86.52**	**89.45**	62.16

The custom-model-B from Table 6 which architecture was discussed in Sect. 3.3, gave the best validation accuracy compared to the custom-model-A for Dataset-A.

Between two custom models, training and validation accuracy for each model in every epoch are recorded to find out the best model that gives comparatively better validation accuracy. From Fig. 3, we can see that after a certain epoch training accuracy highlighted with blue color remains almost the same where validation accuracy highlighted with orange color drops and doesn't increase

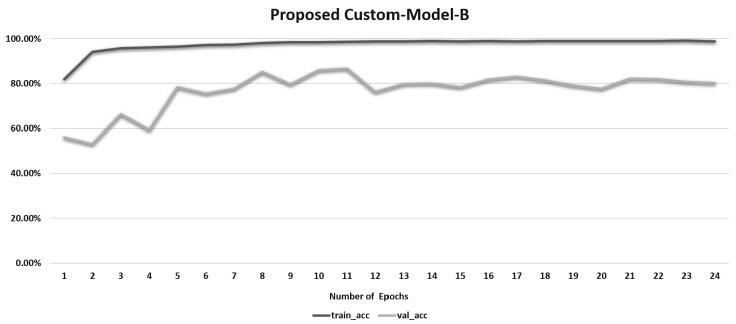

Fig. 3. Illustration of training and validation accuracy of the proposed Custom-Model-B (Color figure online)

prominently. This indicates that there was no need for running the model after that certain epochs. To overcome overfitting some regularization techniques such as **Dropout**, **L2 Regularization** were applied by tuning the hyperparameters which lead to the best performance on the validation set. For this work, 3 different instances of dropout value for custom model-B were considered where dropping 60% of neurons reduces the overall validation loss by an amount of **0.25** that helped to increase the validation accuracy.

4.2 Results from Transfer Learning

Table 7. Results from pre-trained models using DataSet-A

Pre-trained model	No. of total parameters	DataSet-A	
		Training accuracy (%)	Validation accuracy (%)
MobileNetV2	4,297,816	99.88	84.93
NASNetMobile	5,044,012	99.60	85.88
DenseNet121	7,467,480	96.18	76.92
VGG19	29,076,312	84.75	59.93
VGG16	20,024,384	86.50	55.57

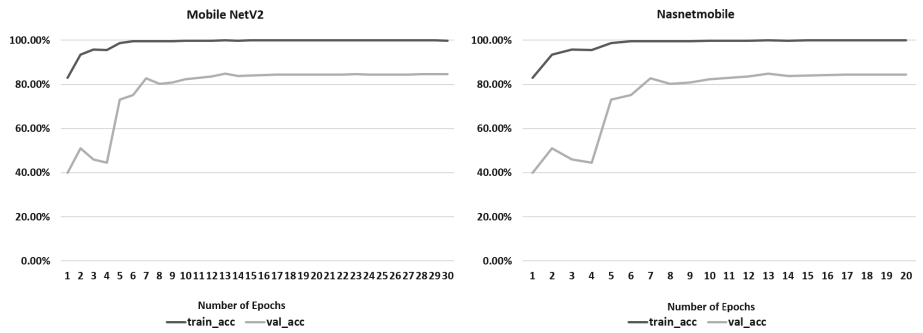

Fig. 4. Illustration of training and validation accuracy of the best two transfer learning models

To improve the validation accuracy, fine-tuning process was introduced where the model was initialized using the technique mentioned in Sect. 3.4. From this configuration, with trainable parameters-9, 051, 928 and non-trainable parameters-20, 024, 384 a validation accuracy of 55.57% was achieved using VGG16 model which weights were pre-trained on **imagenet** dataset and from VGG19 with Trainable parameters-9, 051, 928 & Non-trainable parameters-14, 714, 688, a validation accuracy of 59.93% was achieved where the training accuracy was 84.75%. In both the models, parameters except in fully connected layers were being frozen. As this result was not even close to our custom models, a different technique with other pre-trained models was implied. With this technique, the top layers or fully connected layers of the model was first trained for 10 epochs, then the weights of all the pre-trained layer and the top layer were unfrozen and the same model was trained for the second time. In the first scenario, when the model was only trained with top layers weighs the activation function "Softmax" that relied upon the last fully connected layers trained itself in a way that when in the second time model retrained itself for 25 epochs, it gives much better validation accuracy mentioned in Table 7. From this process using 'MobileNetV2' & 'NASNetMobile' model's pre-trained weights with 2072 and 1176 corresponding neurons, accuracy of **84.93% and 85.88%** were recorded. In the case of Densenet121, VGG16 and VGG19 same configuration could not be applied as there is a huge number of parameters or weights in terms of memory. In case of all the pre-trained models, "MobilenetV" and "NASNetMobile" gives linear growth in terms of validation accuracy. From Fig. 4 we can see that, after running for several epochs, validation accuracy has gone lower for the first 3–4 epochs, then it jumps to 75% and gradually increases to 84% and stabilizes for the remaining epochs. On the other hand, the training accuracy gains 98% accuracy in first 5–6 epoch and remain stable for the rest of the epochs.

4.3 Discussion on Results

The previous work that gave best validation accuracy based on **ASL fingerspelling** dataset was conducted by Pugeault and Bowden [18] where they recorded accuracy on three different instances. They obtained 73% accuracy for using only RGB images, 69% for using only depth information and 75% accuracy for using RGB+depth images. In our work, we have considered only two instances as we only used RGB("DataSet-A") and Depth+RGB("DataSet-B") to measure performances. Although our customized models could not perform better on "DataSet-B" compared to their [18] work but all the other models performed better than [18] on RGB images. A total of 240 unseen color images were used to measure f1 score of both the customized models. Both the models

were asked to measure ground truth values of 10 images from each class. Based on the precision and recall values, f1 score was then generated for each class that is shown in Table 8.

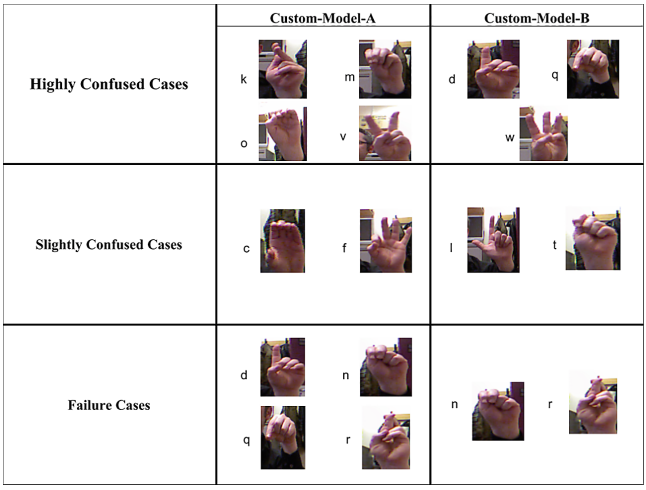

Fig. 5. Illustration of different scenarios of our custom models predictions

For "Custom-Model-A" recall values are significantly higher than the precision values for classes k, m, o, v where for "Custom-Model-B" those classes are d, q, w. The reason behind this might be because signs of c and o, w and f, d and l, m and n, k and r shown in Fig. 5 are quite similar which is why models may get confused while classifying for those particular classes. In case of both the models, the classifiers could not predict n, r out of given images. In case of letter c, f "Custom-Model-A" shows small confusion as the precision values are slightly lower than the recall values for those classes wherein for "Custom-Model-B" those classes are l, t. Although for some classes the custom models could not give accurate predictions overall performance of both the models was good as the macro-average value of "Custom-Model-A" is nearly 59% and for "Custom-Model-B" it is nearly 68%.

Table 8. F1 score obtained from customized models

Class	F1-score		Predicted accurately	
	Custom Model-A	Custom-Model-B	Custom-Model-A	Custom Model-B
a	0.89	1.00	8	10
b	0.89	0.17	8	1
c	0.83	1.00	10	10
d	**0.00**	0.46	**0**	4
e	0.46	0.75	3	9
f	0.91	0.89	10	10
g	0.95	1.00	9	10
h	1.00	0.9	10	8
i	0.84	0.68	8	7
k	0.33	0.95	8	10
l	1.00	0.79	10	7
m	0.62	0.93	10	10
n	**0.00**	**0.00**	**0**	**0**
o	0.12	0.36	1	3
p	0.3	1.00	3	10
q	**0.00**	0.57	**0**	7
r	**0.00**	**0.00**	**0**	**0**
s	1.00	1.00	10	10
t	0.53	0.67	4	5
u	0.95	0.89	9	8
v	0.27	0.74	4	6
w	0.75	0.24	6	2
x	**0.00**	0.71	**0**	7
y	0.83	0.35	10	9

5 Conclusion

In this paper, we present an image-based comparison wise approach to finding models that can interpret sign languages in a much more efficient way from ASL finger Spelling dataset. For that, we have developed two custom models and several transfer learning models based on convolutional neural network. Then for training and validating the network, two approaches were considered in which one approach was to use only RGB images and the other one was to use both RGB and depth information. Our classification results of RGB images exceeded all the previous models. For further improvement, the letters j and z will be included in the video dataset which will be utilized to recognize continuous hand signs.

References

1. Anderson, R., Wiryana, F., Chandra, M., Putra, G.: Sign language recognition application systems for deaf-mute people: a review based on input-process-output. Procedia Comput. Sci. **116**, 441–448 (2017)
2. Arge, F.O.R.L., Mage in CI: Vdcnl-s i r, pp. 1–14 (2015)
3. 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, pp. 1412–1415 (2014)
4. Núñez Fernández, D., Kwolek, B.: Hand posture recognition using convolutional neural network. In: Mendoza, M., Velastín, S. (eds.) CIARP 2017. LNCS, vol. 10657, pp. 441–449. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75193-1_53
5. Ghotkar, A., Kharate, G.K.: Study of vision based hand gesture recognition using Indian sign language (2017)
6. Chollet, F.: Xception: deep learning with depthwise separable convolutions (2014)
7. Hoque, T., Kabir, F.: Automated Bangla sign language translation system: prospects, limitations and applications, pp. 856–862 (2016)
8. Hosoe, H., Sako, S.: Recognition of JSL finger spelling using convolutional neural networks, pp. 85–88 (2017)
9. Huang, G., Weinberger, K.Q.: Densely connected convolutional networks (2016)
10. Karabasi, M., Bhatti, Z., Shah, A.: A model for Real-time recognition and textual representation of Malaysian sign language through image processing. In: 2013 International Conference on Advanced Computer Science Applications and Technologies (2013)
11. Karmokar, B.C., Alam, K.R., Siddiquee, K.: Bangladeshi sign language recognition employing neural network ensemble (2012)
12. Kishore, P.V.V., Kumar, P.R.: Segment, track, extract, recognize and convert sign language videos to voice/text. IJACSA **3**, 35–47 (2012)
13. Koller, O., Forster, J., Ney, H.: Continuous sign language recognition: towards large vocabulary statistical recognition systems handling multiple signers. Comput. Vis. Image Underst. **141**, 108–125 (2015)
14. Kumar, P.K., Prahlad, P., Loh, A.P.: Attention based detection and recognition of hand postures against complex backgrounds (2012)
15. Masood, S., Srivastava, A., Thuwal, H.C., Ahmad, M.: Real-time sign language gesture (word) recognition from video sequences using CNN and RNN. In: Bhateja, V., Coello Coello, C.A., Satapathy, S.C., Pattnaik, P.K. (eds.) Intelligent Engineering Informatics. AISC, vol. 695, pp. 623–632. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-7566-7_63
16. Mekala, P., Gao, Y., Fan, J., Davari, A.: Real-time sign language recognition based on neural network architecture, pp. 195–199 (2011)
17. Prajapati, R., Pandey, V., Jamindar, N., Yadav, N., Phadnis, P.N.: Hand gesture recognition and voice conversion for deaf and dumb. IRJET **5**, 1373–1376 (2018)
18. Pugeault, N., Bowden, R.: Spelling it out: real-time ASL fingerspelling recognition (2011)
19. Rahaman, M.A., Jasim, M., Ali, H.: Real-time computer vision-based Bengali sign language recognition, pp. 192–197 (2014)
20. Rajam, P.S., Balakrishnan, G.: Real time Indian sign language recognition system to aid deaf-dumb people, pp. 1–6 (2011)
21. Rao, G.A., Kishore, P.V.: Selfie video based continuous Indian sign language recognition system. Ain Shams Eng. J. **9**, 1929 (2017)

22. Sandler, M., Zhu, M., Zhmoginov, A., Howard, A., Chen, L.-C.: MobileNetV2: inverted residuals and linear bottlenecks (2018)
23. Savur, C.: Real-time American sign language recognition system by using surface EMG signal, pp. 497–502 (2015)
24. Sarawate, N., Leu, M.C., ÖZ, C.: A real-time American sign language word recognition system based on neural networks and a probabilistic model. Turk. J. Electr. Eng. Comput. Sci. **23**, 2107–2123 (2015)
25. Seth, D., Ghosh, A., Dasgupta, A., Nath, A.: Real time sign language processing system. In: Unal, A., Nayak, M., Mishra, D.K., Singh, D., Joshi, A. (eds.) SmartCom 2016. CCIS, vol. 628, pp. 11–18. Springer, Singapore (2016). https://doi.org/10.1007/978-981-10-3433-6_2
26. Singha, J., Das, K.: Recognition of Indian sign language in live video. Int. J. Comput. Appl. **70**, 17–22 (2013)
27. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: the all convolutional net, pp. 1–14 (2015)
28. Szegedy, C., Vanhoucke, V., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision (2014)
29. Tripathi, K., Baranwal, N., Nandi, G.C.: Continuous Indian sign language gesture recognition and sentence formation. Procedia Comput. Sci. **54**, 523–531 (2015)
30. Uddin, S.J.: Bangla sign language interpretation using image processing (2017)
31. Wazalwar, S., Shrawankar, U.: Interpretation of sign language into English using NLP techniques. J. Inf. Optim. Sci. **38**, 895 (2017)
32. Zoph, B., Shlens, J.: Learning transferable architectures for scalable image recognition (2017)