American Sign Language Posture Understanding with Deep Neural Networks

Md Asif Jalal[a], Ruilong Chen[b], Roger K Moore[a], Lyudmila Mihaylova[b]

[a]Department of Computer Science, University of Sheffield, S1 4DP, UK
[b]Department of Automatic Control and Systems Engineering, University of Sheffield, S1 3JD, UK
{majalal1,rchen3,r.k.moore,l.s.mihaylova}@sheffield.ac.uk

***Abstract*—Sign language is a visually oriented, natural, non-verbal communication medium. Having shared similar linguistic properties with its respective spoken language, it consists of a set of gestures, postures and facial expressions. Though, sign language is a mode of communication between deaf people, most other people do not know sign language interpretations. Therefore, it would be constructive if we can translate the sign postures artificially. In this paper, a capsule-based deep neural network sign posture translator for an American Sign Language (ASL) fingerspelling (posture), has been presented. The performance validation shows that the approach can successfully identify sign language, with accuracy like 99%. Unlike previous neural network approaches, which mainly used fine-tuning and transfer learning from pre-trained models, the developed capsule network architecture does not require a pre-trained model. The framework uses a capsule network with adaptive pooling which is the key to its high accuracy. The framework is not limited to sign language understanding, but it has scope for non-verbal communication in Human-Robot Interaction (HRI) also.**

***Index Terms*—American Sign Language (ASL) Understanding, Neural Network, Capsule Network, Adaptive Pooling**

I. Introduction

Sign language is a visually oriented, natural, non-verbal communication medium, which is used by millions of hearing impaired people around the globe as their first language. According to the British Deaf Association, there are 151,000 people use sign language [1]. The main two components of sign language are finger-spelling (postures) and dynamic hand movement (gestures) [2]. Hearing-impaired people often find it quite challenging to communicate with other people because most do not know sign language. Therefore, an artificial sign language translator would be useful in reducing the communication barrier.

Although, there has been extensive research on ASL detection, it still remains a relevant research field due to the unavailability of an accurate method. There are two main approaches [3, 4]: using gloves with sensors to detect joint movements and using vision [5].

Sensor-based methods with Bayesian networks and neural network classifiers were popular in the early 2000s [3, 4, 6–8]. Wearable sensor gloves are used for getting the relative motion of the fingers and hands to get the kinematic parameters to predict the sign language. Cheap wearable technologies are proposed in [6–8]. Using colored gloves and constrained grammars reported low error rates on the training data (0.6%) and an independent test data (3%) [9, 10].

Linear classifiers are also widely used for detecting postures and gestures because they are relatively simple models compared to Bayesian models and they get high accuracies (96%) also [11–13]. Hidden Markov models and Bayesian models also achieved higher accuracies [14, 15]. The problems with those approaches arise from hand-coded feature extraction with heavy pre-processing and the constrained experimental environments [16].

Neural networks have an advantage over these networks because they learn essential features to classify the data [17]. Feed-forward neural networks also need image processing and hand-coded feature extraction. Convolutional Neural Networks (CNNs) have been very useful for understanding gestures and events [18]. The most relevant works to date are by Garcia et al. [16] and Pigou et al. [5]. They used CNNs for classification. Convolutional layers work as filters and do not require hand-coded feature inputs. CNNs also have some fundamental defects. They work as filters but do not preserve all the spatial-temporal features of an image.

Sign language understanding can also be seen as a gesture classification task. Bheda et al. [19] used Deep Convolutional Neural Network (DCNN) to address using a gesture classification approach. The depth and color of an image are also used in some research. Ameen et al. use a CNN to classify American sign language (ASL) using the depth and colour of images and they report 82% precision and 80% recall in their experiments [20].

The model contains three convolutional layers, two spatial adaptive pooling layers, one 6D convolutional capsule layer and one fully connected capsule layer.

This paper aims to present an efficient framework for sign language posture understanding. Using adaptive layer pooling allows the network to be trained with variable sized images as the produce fixed length matrices. Networks trained with multiple sized images could enhance scale-invariance. Those feature matrices are fed into the capsule layer. Capsule layers provide transitional invariance and robustness to the framework. The images in the dataset are rotated slightly, making it quite challenging.

The rest of the paper is organized as follows. In Section II, the deep learning approaches, which are used in the proposed framework, are discussed. Section III explains the proposed deep framework. Simulation results and discussions are presented in Section IV and Section V summarizes the work.

II. Deep Learning Approaches

A. Convolutional Neural Network

Convolution is a mathematical operation operated on two functions to produce a new function that expresses the amount of overlap of one function shifted over another function. If $f(x)$ and $g(x)$ are two functions over a continuous variable x, the convolution over an infinite interval would be

$$f(\mathbf{x}) * g(\mathbf{x}) = \int_{-\infty}^{+\infty} f(\Gamma) \times g(\mathbf{x} - \Gamma) d\Gamma, \tag{1}$$

where Γ is continuous time step, $*$ is the convolution operator and $\times$ is ordinary multiplication. If f and g are functions over discrete variables, with k being a discrete time, then convolution of g over f will be defined as

$$y[\mathbf{x}] = f[\mathbf{x}] * g[\mathbf{x}] = \sum_{k=-\infty}^{+\infty} f[\mathbf{k}] \times g[\mathbf{x} - \mathbf{k}], \tag{2}$$

where $y[x]$ is the resulting output, $f[x]$ is the input and $g[x]$ is the filter. In signal processing, $g[x]$ is the impulse function over $f[x]$. For functions over two discrete variables x and y the convolution would be

$$y[\mathbf{x}, \mathbf{y}] = f[\mathbf{x}, \mathbf{y}] * g[\mathbf{x}, \mathbf{y}] = \left(\sum_{n_1=-\infty}^{+\infty} \sum_{n_2=-\infty}^{+\infty} f[\mathbf{n_1}, \mathbf{n_2}] \times g[\mathbf{x} - \mathbf{n_1}, \mathbf{y} - \mathbf{n_2}] \right), \tag{3}$$

where n_1 and n_2 are discrete timestamps. In a linear time invariant system the output $y[x]$ could be seen as the combination of convolution operations of kernel $g[x]$ on input $f[x]$. In image processing, the interval is finite. So, the equation (3) will be converted to

$$y[\mathbf{x_c}, \mathbf{y_c}] = f[\mathbf{x_a}, \mathbf{y_a}] * g[\mathbf{x_b}, \mathbf{y_b}] = \left(\sum_{n_1=0}^{x_a-1} \sum_{n_2=0}^{y_a-1} f[\mathbf{n_1}, \mathbf{n_2}].g[\mathbf{x} - \mathbf{n_1}, \mathbf{y} - \mathbf{n_2}] \right), \tag{4}$$

where $0 < c < a{+}b{-}1$. The $g[x_b, y_b]$ could be perceived as the local receptive field, sharing the same set of weights, working on different parts of the input image $f[x_a, y_a]$, in which the neurons extract visual features (edges, corners or more abstract features) and combine the set of outputs to form feature maps. Kernels of size $[x \times y \times N]$ ([height $\times$ width $\times$ depth], and $n = 1, 2, \cdots, N$) are used, the n^{th} convolutional feature map can be denoted as:

$$y_n = f\left(\sum_j g_n * x_j \right), \tag{5}$$

where g_n is the n^{th} kernel and x_j ($j = 1, 2, \cdots, J$) is the j^{th} input feature map of size $[A \times B]$ and $f(\cdot)$ is a nonlinear activation function because the output of convolution is linear. To make it adaptable to a more complex problem space, a non-linear activation function is applied. In this paper, the Rectified Linear Unit (ReLU) is applied with convolution layers [21, 22]. ReLU is formally defined as:

$$f(x) = \begin{cases} x, & \text{if } x >= 0. \\ 0, & \text{if } x < 0. \end{cases} \tag{6}$$

Generally, CNNs contain several of these convolution layers along with spatial or temporal sub-sampling [23].

B. Adaptive Pooling

Pooling or subsampling plays a significant role in CNNs. Although, the feature map loses some of its spatial features for pooling operation, it has been proven to be useful for CNNs [24, 25]. The pooled vector is given by $y = f(y_1, y_2, ..y_n)$, where $y_i \in \mathbb{R}^d (i = 1, 2, ..., N)$, d is the dimensionality and n is the number of feature descriptors. The $f(.)$ function is the pooling operation. In this paper, a spatial adaptive max-pooling method is used [26].

Spatial pooling preserves the spatial features by pooling in local spatial pooling bins. The size of spatial bins is proportional to the input image size. The output of each filter from the previous convolution layer is stored in each of the spatial bins using a max pooling method. If there are n filters in the last convolution layer and number of spatial bins are m, the output of the pooling layer will be n, m dimensional vectors. Adaptive spatial pooling is efficient in image classification, object detection problems with higher accuracies and keeping more spatial properties of an image [27, 28]. Also, it provides the network with the ability to be trained with multiple size images.

C. The Capsule Neural Network

Unlike the connections between adjacent convolutional layers (which are through neurons in a CNN) adjacent capsules layers are connected by capsules in a capsule network. For each capsule (represented as a vector), the output of the capsule j can be denoted as:

$$\mathbf{v}_j = g(\mathbf{s}_j), \tag{7}$$

where v_j is the output vector of capsule j, $\mathbf{s}_j$ is the input to the capsule j and $g(\cdot)$ is a squashing function given by:

$$g(\mathbf{x}) = \frac{||\mathbf{x}||^2}{1+||\mathbf{x}||^2}\frac{\mathbf{x}}{||\mathbf{x}||}, \tag{8}$$

with $\mathbf{x}$ is the input vector of the function.

The squashing function makes the length of a short vector shrink close to 0, and a long vectors shrink close to 1. The length of a capsule is used to represent the existence probability of the corresponding entity or part of an entity. Parameters in each capsule represent various properties such as position, scale and orientation of a particular entity [29].

Apart from the capsules in the primary capsule layer, the total input of the capsule $\mathbf{s}_j$ can be calculated by:

$$\mathbf{s}_j = \sum_i c_{ij}\mathbf{o}_{j|i}, \tag{9}$$

where $\mathbf{o}_{j|i}$ is the predicted output of the capsule j (in the current capsule layer) made by the capsule i from the previous capsule layer. The coefficients c_{ij} are coupling coefficients determined by a routing algorithm

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})}, \tag{10}$$

where b_{ij} denote that the log prior probabilities of the capsule i are coupled with the capsule j from the previous capsule layer to the current capsule layer. The term k is an index though all capsules in the current layer. The coefficients b_{ij} are initialized to zero and their values are updated by a routing algorithm.

In the routing algorithm, b_{ij} is updated by:

$$b_{ij}^{(r+1)} = b_{ij}^{(r)} + \mathbf{v}_j \cdot \mathbf{o}_{j|i}, \tag{11}$$

where r is the iteration index. The scalar product of $\mathbf{v}_j$ and $\mathbf{o}_{j|i}$ is the cosine similarity measurement. The term $j|i$ means from i^{th} capsule to j^{th} capsule. If the two vectors are similar, the b_{ij} will be updated with a large step. On the contrary, if the two vectors are different, b_{ij} will be updated with a small step. Intuitively, a lower level capsule i predicts the output of capsule j. The probability that the capsule i and the capsule j are coupled is determined by the product of the prediction and the actual output.

In equations (9) and (11), the predictions $\mathbf{o}_{j|i}$ are given by:

$$\mathbf{o}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i, \tag{12}$$

where $\mathbf{u}_i$ is output of the capsule i in the previous capsule layer and $\mathbf{W}_{ij}$ are transformation matrices connecting capsules between the previous capsule and the current capsule layer. The final capsule layer connects the classification labels directly. There are D (D is the number of classes) capsules, with the length of each capsule represents the prior probability of the classification object [30]. A margin loss is applied in capsule network in order to allow multiple classes to exist in the same image. The loss L_d for the class $d = 1, 2, \cdots, D$ is given by

$$\begin{aligned} L_d = T_d\ max(0, m^+ - ||\mathbf{v}_d||)^2 \\ + \lambda(1-T_d)\ max(0, ||\mathbf{v}_d|| - m^-)^2, \end{aligned} \tag{13}$$

where $T_d = 1$ if and only if a digit of class d exists and $||\mathbf{v}_d||$ represents the length of vector $\mathbf{v}_d$. For the correct class d, loss starts to accumulate when the length of the $\mathbf{v}_d$ is under m^+. Meanwhile, for the unrelated class d, loss begins to accumulate when the length of the $\mathbf{v}_d$ is beyond m^-. The λ is a controlling parameter and the total loss $L_{total} = \sum_d L_d$, which simply sums the losses from all the final layer capsules.

In the capsule network, the back-prorogation is applied to update the weights in the convolutional kernels and the transformation matrices. The routing phase is applied to update the weights for the coupling process coefficients c and the log prior probabilities b [30]. The vector to vector transformation could potentially extract more robust features than scale to scale transformation in a CNN [29].

III. The proposed framework

The general architecture of the proposed framework is given in Figure 1. The first convolution layer has 64, $[11 \times 11]$ sized ([height $\times$ width] convolution kernels. The convolution process is applied with a stride of 1, padding 2 and ReLU activation function. A spatial adaptive max pooling layer follows this layer. Neural networks with fixed filter size extract differently sized features for different sized images. Fully connected layers need fixed length input. If the feature scale is different due to different sized images, it would be a problem. Hence, the advantage of using this adaptive pooling layer is, it can take an input of variable size, arbitrary aspect ratio, scales and produce fixed sized output. Therefore, this method allows flexibility in the network. The first adaptive pooling layer produces 64 $[25 \times 25]$ neurons. After this, two convolution layers are stacked together having 192, $[5\times5]$ and 256, $[3\times3]$ ([height $\times$ width]) kernels respectively with padding 1 and ReLU activation function. An adaptive pooling layer follows producing 256 channels of fixed output neurons $[20 \times 20]$.

On the top of the second pooling layer, the primary capsule layer has thirty-two channels of six convolutional units with the kernel size of $[9\times9]$ and stride 2. Thirty-two channels with

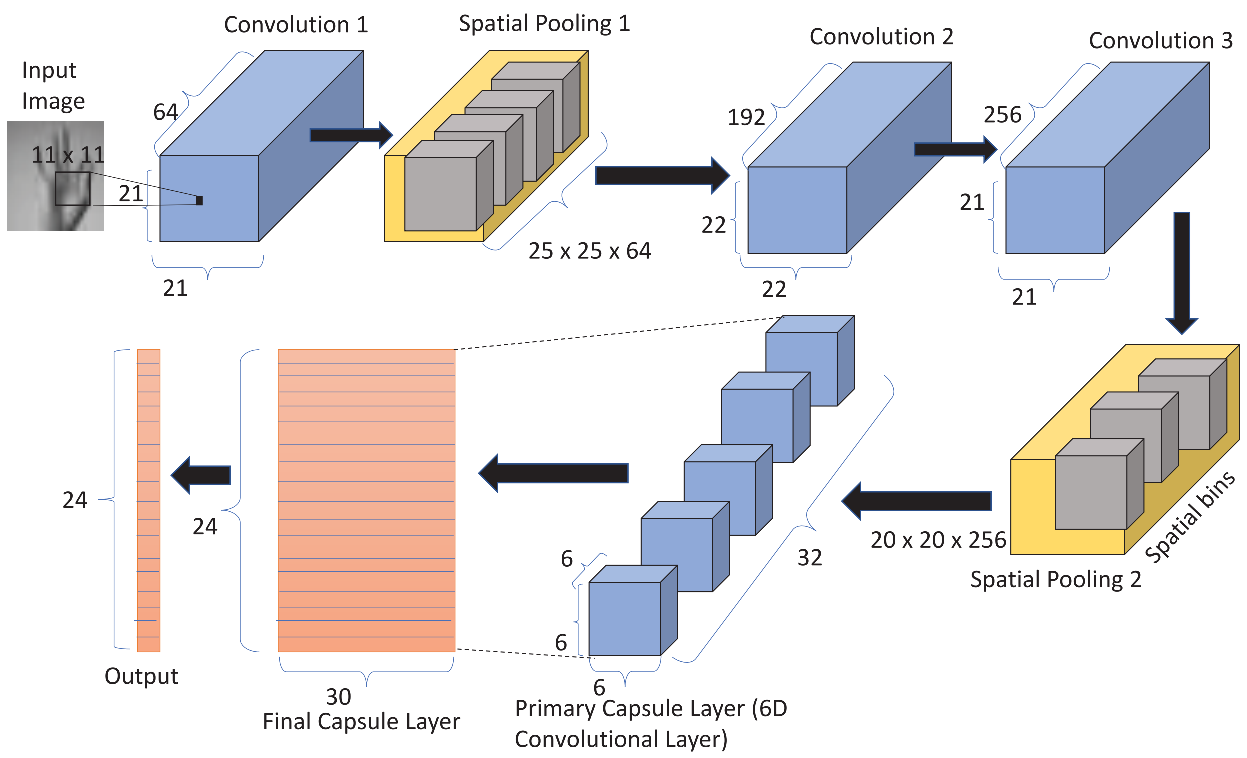

Fig. 1: The proposed framework consisting of a convolutional layer and spatial adaptive pooling with capsule network routing.

the block size of [$6 \times 6 \times 6$] are generated. Each channel contains $6{\times}6{=}36$ capsules, with each capsule containing vector of length six. The detailed process of capsule generation can be found in our work [30]. Each capsule in the convolutional capsule layer shares their weights with each others.

Transformation matrices connect adjacent capsule layers. The convolutional capsule layer has [$32 \times 6 \times 6$] sized capsule outputs. A routing process is applied in the capsule layer according to the work of Sabour et al. [29]. The final capsule layer gives 24 output classes with 30 length vectors per class. Those are passed through a softmax function, and the class probabilities are computed. We have used PyTorch to implement the model and specifically used ‘AdaptiveMaxPooling’ library function for the pooling.

Those class probabilities are used to compute the loss at each training case. They are also used as a mask to nullify the unwanted output vector classes. The remaining output vector is used to reconstruct the image through decoder network. The reconstruction loss is also used as a regularizing expression to add more robustness in the network [29].

IV. Performance Evaluation

The capsule network from Sabour et al. [29] is used as the benchmark for this paper. A deeper version of that capsule network has also been designed with four capsule layers to compare with the proposed framework. The deep capsule net contains one simple convolution layer for feature map extraction and one convolutional capsule layer and three fully connected capsule layers. When capsule layers are stacked together, they are one of the most computationally costly frameworks. After using different routing iterations, it has been observed that three rotations are efficient enough to get quick convergence with reasonable computational complexity. All of these results presented in the paper are based on three routing iterations.

A CNN model has also been used to measure the performance of the proposed framework. Alexnet [25] is an efficient CNN model. We have changed a few parameters to accommodate it with out dataset (image size $28{\times}28$ pixels). We call it ConvNet in Figure (5) and in the discussion of the results.

The performance evaluation of the networks are conducted in Python using PyTorch library. The machine on which the experiments have been performed has an Intel core i5-3210M CPU @ 2.50GHz $\times$ 4 , 8 GB Primary memory and one Nvidia GeForce GTX 1070 (extended GPU).

A. Dataset

The ‘Kaggle’ American Sign Language Letter” database of hand gestures has been used to evaluate the framework. The American Sign Language database has 24 classes of letters. The database contains all the letters except ‘J’ and

'Z' because they do not have static postures. The training consists of 27455 items and test set has 7172 items. The dataset contains gray scale 28×28 pixel images. According to the dataset description, the images have been modified with at least 50+ variations. For example, 5% random pixelation, +/- 15% brightness/contrast, three degrees rotation etc. These modifications also altered its image resolution and made this dataset a challenging one.

B. Performance of the proposed capsule network with reconstruction loss

According to ([29]) the reconstruction loss is considered while calculating the total loss in each training batch. Figure 2 illustrates the performance of the proposed method in terms of testing accuracies. The testing accuracies are evaluated in each training epoch with new test data.

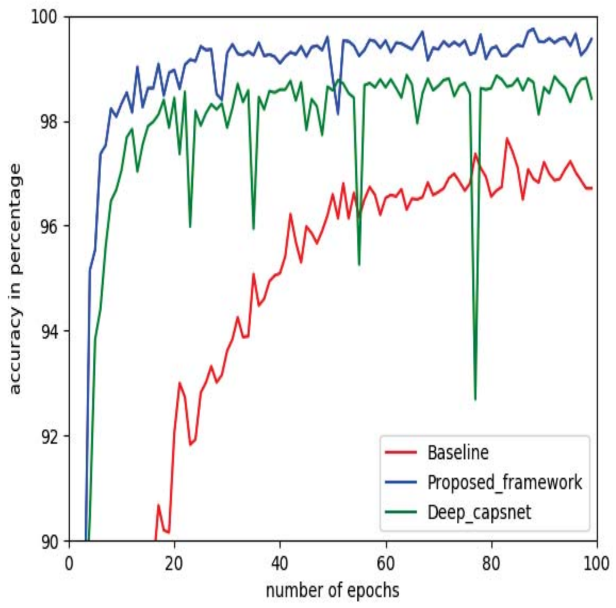

Fig. 2: The accuracy (%) of the proposed framework, baseline capsulenet and deep capsulenet on the testing data while considering reconstruction loss.

The performance of the baseline capsulenet and the deep capsulenet are also shown in figure 2. Some observations can be made from the figure. The rate of convergence is higher in the proposed framework than the baseline two layer capsulenet. The 4-layer deep capsulenet is closer to the proposed network in terms of convergence rate, but it is less efficient in terms of robustness. The maximum accuracy in the baseline is 97.65%. The maximum accuracy of the proposed framework is 99.74%.

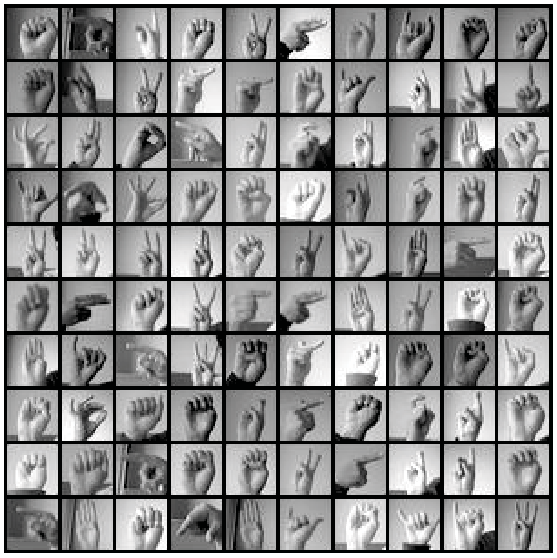

Fig. 3: Ground truth images

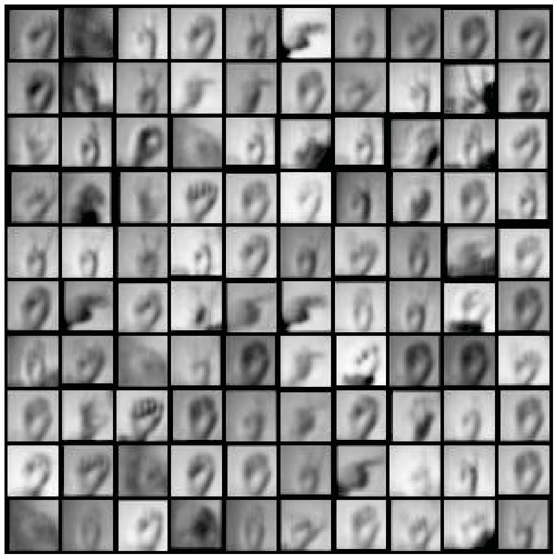

Fig. 4: Reconstructed images

C. Performance of the proposed capsule network without reconstruction loss

In this section, the loss function is calculated without the image reconstruction loss. Figure 5 represents the performance of different frameworks in each training epoch, up to 100 epochs. The baseline method has higher maximum accuracy but stability and consistency wise the proposed method works slightly better. The 4-layer deep Capsulenet is more stable and robust. The test accuracies dropped a significant amount for the baseline method and for the convnet. One of the reasons could be the absence of the regularizing term while calculating the loss (for example, reconstruction loss). There are fewer capsules in the proposed method, which also could be the

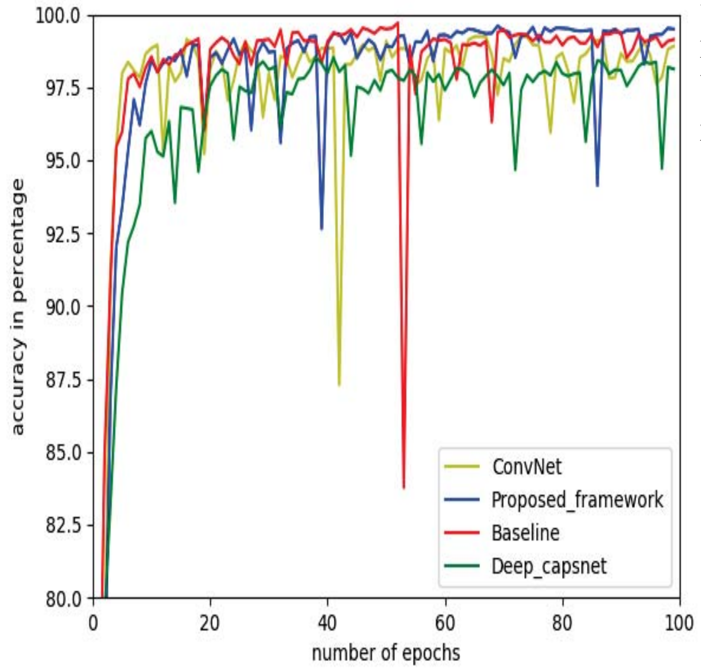

Fig. 5: The accuracy (%) of the proposed framework, deep capsulenet, the baseline method and the convnet on testing data without considering reconstruction loss.

Method	Maximum Test Accuracy (%)		Minimum Test error (%)	
	With Reconstruction	Without Reconstruction	With Reconstruction	Without Reconstruction
BaseLine	97.65	99.70	0.652	0.633
Deep Capsulenet	98.87	98.54	0.639	0.634
ConvNet	–	99.26	–	–
Proposed Framework	**99.74**	**99.60**	**0.641**	**0.632**

Table I: Summery of experiment results

reason.

Table I gives a summary of all the experiments. Clearly, it can be seen that, for this dataset, the proposed method is an improvement of the 4-layer capsule net and baseline 2-layer capsulenet. Even without the reconstruction loss calculation, it has higher accuracy than the other two frameworks.

V. Conclusion

In this paper, a posture learning framework has been proposed for sign language recognition. Although, the image quality of the dataset is not very high, the framework shows good results. This framework is robust for rotation and images quality. The concept of capsules and pooling are used simultaneously in the network. Originally, Capsules Network's routing by agreement came as an alternative to pooling methods [29]. This research confirms that using both pooling and capsules routing on the same network can improve the networks accuracy and convergence speed. The adaptive pooling used in this framework allows the network to train with multiple size images, which adds scale-invariance and prevents the network from over-fitting. Future extension of this work would be upgrading this framework to non-static gestures also. One of the many possible ways to do this would be to use recurrent neural network layers with the capsules.

References

[1] B. D. Association, "Bsl statistics. in: Sign language week 2018."

[2] H. Lahamy and D. D. Lichti, "Towards real-time and rotation-invariant American sign language alphabet recognition using a range camera," *Sensors (Switzerland)*, vol. 12, no. 11, pp. 14 416–14 441, 2012.

[3] S. A. K. Mehdi Y. N., "Sign language recognition using sensor gloves," *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP '02.*, vol. 5, pp. 2204–2206 vol.5, 2002.

[4] S. Sidney Fels and G. E. Hinton, "Glove-Talk: A Neural Network Interface Between a Data-Glove and a Speech Synthesizer," *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 2–8, 1993.

[5] L. Pigou, S. Dieleman, P.-J. Kindermans, and B. Schrauwen, "Sign Language Recognition Using Convolutional Neural Networks," in *Computer Vision - ECCV 2014 Workshops: Zurich, Switzerland, September 6-7 and 12, 2014, Proceedings, Part I*, 2015, pp. 572–578.

[6] A. K. Singh, B. P. John, S. R. Venkata Subramanian, A. Sathish Kumar, and B. B. Nair, "A low-cost wearable Indian sign language interpretation system," in *International Conference on Robotics and Automation for Humanitarian Applications, RAHA 2016 - Conference Proceedings*, 2017.

[7] C.-H. Chuan, E. Regina, and C. Guardino, "American Sign Language Recognition Using Leap Motion Sensor," in *Proc. from the 2014 13th International Conference on Machine Learning and Applications*, 2014, pp. 541–544.

[8] S. Vutinuntakasame, V. R. Jaijongrak, and S. Thiemjarus, "An assistive body sensor network glove for speech- and hearing-impaired disabilities," in *Proceedings - 2011 International Conference on Body Sensor Networks, BSN 2011*, 2011, pp. 7–12.

[9] T. E. Starner and A. Pentland, "Visual Recognition of American Sign Language Using Hidden Markov Models," *Media*, pp. 189–194, 1995.

[10] T. Eugene Starner, "Visual Recognition of American Sign Language Using Hidden Markov Models," *Massachusetts Institute of Techonology Cambrige*, vol. 298, pp. 1–71, 1995.

[11] J. Singha and K. Das, "Indian Sign Language Recognition Using Eigen Value Weighted Euclidean Distance Based Classification Technique," *arXiv preprint arXiv:1303.0634*, vol. 4, no. 2, pp. 188–195, 2013. [Online]. Available: http://arxiv.org/abs/1303.0634

[12] D. Aryanie and Y. Heryadi, "American sign language-based finger-spelling recognition using k-Nearest Neighbors classifier," in *2015 3rd International Conference on Information and Communication Technology, ICoICT 2015*, 2015, pp. 533–536.

[13] R. Sharma, Y. Nemani, S. Kumar, L. Kane, and P. Khanna, "Recognition of Single Handed Sign Language Gestures using Contour Tracing Descriptor," *World Congress on Engineering*, vol. 2, pp. 754–758, 2013.

[14] T. Starner and A. S. Pentland, “Real-Time American Sign Language Recognition Hidden Markov Models from Video Using,” *AAAI Technical Report FS-96-05*, pp. 109–116, 1996.

[15] M. Jebali, P. Dalle, and M. Jemni, “Hmm-based method to overcome spatiotemporal sign language recognition issues,” in *2013 International Conference on Electrical Engineering and Software Applications, ICEESA 2013*, 2013.

[16] B. Garcia and S. a. Viesca, “Real-time American Sign Language Recognition with Convolutional Neural Networks,” *Convolutional Neural Networks for Visual Recognition*, 2016.

[17] Y. F. Admasu and K. Raimond, “Ethiopian sign language recognition using Artificial Neural Network,” in *Intelligent Systems Design and Applications ISDA 2010 10th International Conference on*, 2010, pp. 995–1000.

[18] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” in *Proceedings of the IEEE*, vol. 105, no. 12, 2017, pp. 2295–2329.

[19] V. Bheda and D. Radpour, “Using Deep Convolutional Networks for Gesture Recognition in American Sign Language,” *CoRR*, vol. abs/1710.06836, 2017.

[20] S. Ameen and S. Vadera, “A convolutional neural network to classify American Sign Language fingerspelling from depth and colour images,” *Expert Systems*, vol. 34, no. 3, 2017.

[21] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” *Proceedings of the 27th International Conference on Machine Learning*, no. 3, pp. 807–814, 2010.

[22] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical Evaluation of Rectified Activations in Convolution Network,” *ICML Deep Learning Workshop*, pp. 1–5, 2015.

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

[24] H. Wu and X. Gu, “Max-pooling dropout for regularization of convolutional neural networks,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9489, 2015, pp. 46–54.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Advances In Neural Information Processing Systems*, pp. 1–9, 2012.

[26] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[27] Y. Liu, Y. M. Zhang, X. Y. Zhang, and C. L. Liu, “Adaptive spatial pooling for image classification,” *Pattern Recognition*, vol. 55, pp. 58–67, 2016.

[28] Y. H. Tsai, O. C. Hamsici, and M. H. Yang, “Adaptive region pooling for object detection,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June-2015, 2015, pp. 731–739.

[29] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” in *Advances in Neural Information Processing Systems*, 2017, pp. 3859–3869.

[30] R. Chen, M. Jalal, L. Mihaylova, and R. Moore, “Learning capsules for vehicle logo recognition,” in *Submitted to 21th International Conference on Information Fusion*, 2018.