SIGN LANGUAGE ALPHABET RECOGNITION USING ANN AND CNN

Mayand Kumar
Department of Software Engineering
Delhi Technological University
Delhi,India
mayandkumar10@gmail.com

Janvi Arora
Department of Software Engineering
Delhi Technological University
Delhi, India
janviarora1108@gmail.com

Abstract— **Sign Language is an important part of the life of those who have speaking and hearing disabilities. Recognition of American Sign Language is very challenging using Computer Vision as Sign Language are very much complex and have high intraclass variations. In this paper, Convolutional Neural Networks (CNN) is used to recognize ASL Alphabets. This algorithm is useful for recognition as a deep network is expected for the ASL Alphabet Classification task. Pre-Processing steps of the MNIST dataset are done in the first phase. After the first phase, Various important features of pre-processed hand gesture image are computed. In the final phase, depending on the properties computed or calculated in the initial phases, the Accuracy and AUC Score of the network model with which it can recognize the sign language Alphabets were found. The proposed CNN network was able to achieve an AUC Score of 0.9981 and an accuracy of 0.9963.**

Keywords— Convolution neural network (CNN), Artificial Neural Network (ANN) American sign language (ASL), ASL number, Data Augmentation, Sign language recognition component.

I. INTRODUCTION

The Sign Language Alphabets are created using hand and facial gestures to express the thoughts of people with speaking and hearing disabilities to standard people who do not have speaking or hearing disabilities. Most of the population may not purely understand Sign language. This results in a vast communication gap between normal people and the person with disabilities. There is certain technological development to help people speak and hearing disabled people as it is impossible for any human to be with the disabled person all the time and help them in their daily activities as a translator. An approach can be thought of that can clarify sign gestures into decipherable text with the help of technology so that it is understandable by the normal public. This approach will surely reduce the communication gap and will make the communication between impaired and standard people more feasible. There more than 100 different sign languages across the world used by deaf and speech impaired groups, such as Indian Sign Language, American Sign Language (ASL), Italian Sign Language, and many others. More than thousands and millions of people globally and about millions of people in India are using Sign Language as their main mode of communication.

American Sign Language is the sign language that is widely used and it ranks fourth most used language in North America. It is a surprise to know that not only in the USA but ASL is used in more than 30 other nations where English is the main language of communication. Around a million people in the USA and various parts of the globe use ASL as their primary mode of communication [1]. ASL is a quite vast and complex language that is created using hand, finger actions, and facial gestures to express the thoughts of speech-impaired people. ASL Language is recognized as appropriate and original language, It contains a lot of variations like any other language contains, For example: Italian and German. Sign Language is a magnificent piece of art of interaction that is favourable to a wide community of speaking and hearing disabled people. The existence of ASL promotes happiness and hopes amongst impaired people and its wide impact is marvellous and eye-catching [2].

ASL is a set of 26 gesture signs known as an American Manual Alphabet that can be used to express various words of English Dictionary available. There are 19 different hand shapes made by people to communicate in ASL to form 26 American Manual Alphabets. Since there are fewer hand shapes available so some hand shapes convey different alphabets if we change the orientation of that hand shape. For Example: ‘K’ and ‘P’ letters. Hand Gestures are also available to express numbers from ‘0’ to ‘9’. Though there are no hand gestures for specific nouns and specific terms [3], there various facial and hand gesture signs to express various English words. ASL’s set of 10 numbers that’s is from 0 to 9 and 26 alphabets gesture signs of English Alphabets from A to Z are shown in Figure 1.

American Sign Language or ASL is a normal language and has mostly the same characteristics as a spoken language. ASL is used as a mode of conversation using hand and facial gestures as stated earlier. It is one of the main modes of communication of most North Americans having speaking and hearing disabilities and it is used by normal people too. Sign language

recognition will make communication easier if one person isn't well-versed with the language.

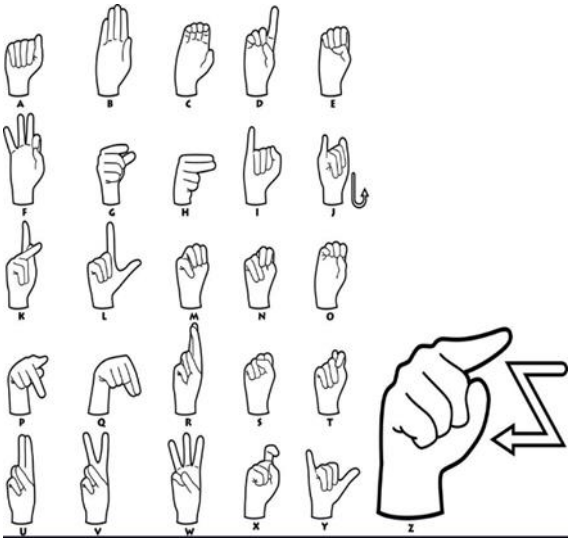

Fig. 1. Poses of Hands for each letter of the English Alphabet.

II. LITERATURE SURVEY

Many ways have been put forward to accomplish the problem of recognizing sign language gestures.

Some prior work involves [4] the use of SVM. SVM was earlier used to classify gestures in SASL [5] and, parallel hidden Markov models were used as well.

An approach to classify Sign language is proposed. The model proposed recognizes ASL alphabets gestures with a success rate of 86.67%.

A real-time interaction system between humans and computers based on hand gestures is developed. In this model, images were pre-processed by applying various operations like hand color filtering, morphological transformations, etc. Later the CNN was used on the dataset. A Kalman estimator was used so that whenever a gesture is identified the mouse pointer will move in response to it. Lastly, the system was able to achieve 99.8 % of accuracy on 16 gestures.

Singha made a dataset that contains 240 images of 24 signs present in Indian Sign Language or ISL. Then the Eigenvalues were extracted from the images in the dataset and were classified based on their Euclidean Distance. This system was able to achieve 97% accuracy.

III. METHODOLOGY

Collecting data is one of the most essential and vital parts of any exploration. It is essential to collect data that is relevant and satisfies the requirements needed for the research. The data should contain readings that replicate all scenarios, including extreme cases. This helps in having more thorough and realistic observations. The data collected forms the foundation for any research and should be collected with utmost care.

For this research, we decided to design the dataset to match the MNIST classic dataset. Therefore, in the dataset, we have labeled each element as training or test cases ranging from 0 to 25, which act as a matched map to all the alphabets in the English language ranging from A-Z. The dataset contains 27,455 training samples and 7172 testing data. This is approximately half the size of the MNSIT dataset, but it replicates it accordingly as every header row has a label of pixel1 to picxel784, which represents the grayscale values of 28x28 pixel images in which the values range from 0-255[13]. The data collected in the MNSIT data is varied by having the same signs made by different subjects in different backgrounds. Moreover, the images were also cropped in various ways; however, the desired hand region was left unchanged.

After exploring the dataset, as shown in figure 2, the data is observed to be evenly balanced across the values. The wide variety of training sample values are evenly distributed, as can be seen from the visualization.

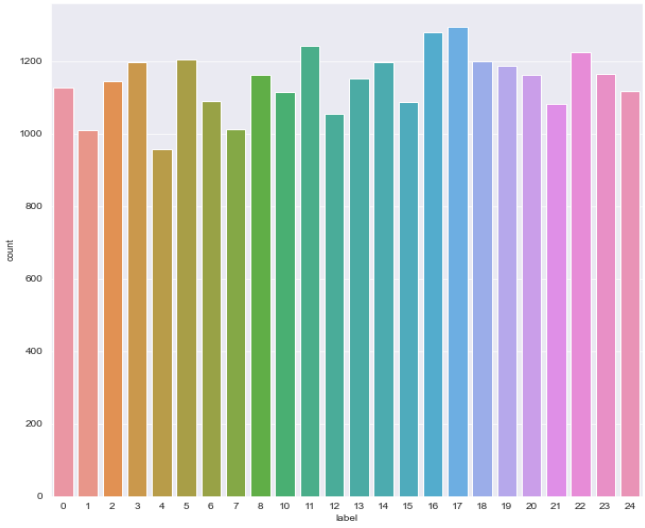

Fig. 2. Number of training examples for each label

A) Loading and Pre-processing of Dataset

We started by importing the dataset to work upon and making it suitable to perform various algorithms to predict sign language alphabet labels correctly. In the given dataset, we already have separate training and test. We visualized our dataset, and it came out to be balanced. We also store the labels of each image, which is mapping with their respective labels in a list, and generates a random image to perform class label verification Figure 3. We performed a grayscale normalization to reduce the effect of illumination's differences. Moreover, the CNN converges faster on [0..1] data than on [0..255] Figure 4. After applying normalization, we split the training set into parts, which is the training set and validation set. So now, we have three sets of data that is training set, validation set, and

test set. In this particular problem, each of the 27455 images of the input has 784 columns. To feed this data into the CNN model, we converted the single-dimensional data into CNN accepted format of 3 dimensions. Thus, we accomplished this by converting each row of 784 columns into (28,28,1) matrix.

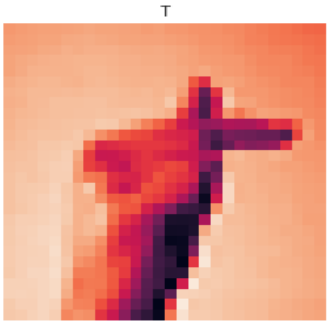

Fig.3 A Random image for class label verification.

Fig. 4. Grayscale images

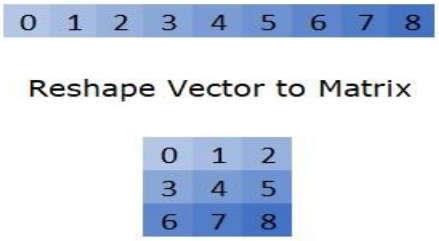

Fig. 5. Reshaping of vector to a matrix.

B) Data Augmentation

In Data augmentation from one image, we generate more images. Therefore, we can have a variety in our dataset, and it is also used to increase the size of the dataset to avoid overfitting. Our data augmentation includes random rotation ranging 10 degrees and random height and width shift in the range of 0.1, random zooming, and random flips. This, we implemented using Keras image pre-processing module.

C) Model Building

We build the model using the combination of convolution, max pooling, and flattening layers in Keras. We also used batch normalization and dropout layers to prevent the model from overfitting the data. We used a callback to determine if our validation set Accuracy did not increase even after 2 (patience level) consecutive epochs during the model fitting stage. Then, we will alter our learning rate by a factor of 0.5.

D) ANN Model

We are using keras which is an open-source library to build our model. So, initially we imported keras and all the other dependencies. Then, we have initialized our model as sequential model. Input layer of our network expects rows of data with 784 variables .We have input dimension as 784 which is equal to the number of columns in the dataset . We have used four hidden layers each of which has 404 nodes and uses the Rectified Linear Unit (ReLU) Activation Function because of its fast computation capabilities. 'Dense' at each layer signifies that layers will be fully connected in our model . We have also added dropout between the hidden layers for prevention of overfitting. Along with it ,we have also added batch normalization between the layers for faster working of our model. Batch Normalization adds few new layers in our neural network which normalizes the values corresponding to input layer.

The total trainable parameters to be calculated are 820,549 as seen in Figure 6., representing each layer with its output shape and trainable parameters.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 404)               317140
_________________________________________________________________
batch_normalization_3 (Batch (None, 404)               1616
_________________________________________________________________
dropout_3 (Dropout)          (None, 404)               0
_________________________________________________________________
dense_6 (Dense)              (None, 404)               163620
_________________________________________________________________
batch_normalization_4 (Batch (None, 404)               1616
_________________________________________________________________
dropout_4 (Dropout)          (None, 404)               0
_________________________________________________________________
dense_7 (Dense)              (None, 404)               163620
_________________________________________________________________
batch_normalization_5 (Batch (None, 404)               1616
_________________________________________________________________
dropout_5 (Dropout)          (None, 404)               0
_________________________________________________________________
dense_8 (Dense)              (None, 404)               163620
_________________________________________________________________
dense_9 (Dense)              (None, 25)                10125
=================================================================
Total params: 822,973
Trainable params: 820,549
Non-trainable params: 2,424
_________________________________________________________________
```

Fig. 6. Trainable Parameters

At output layer , we have 25 nodes and we have used Softmax activation function .We have used sparse categorical cross-entropy because we have labelled as targets. Finally, we compile our model, and for minimization of cost function , we use 'adam' optimizer (stochastic Gradient Descent Method) to attain a global minimum. 'Adam' optimizer is used with a learning rate of 0.001 and values of b1=0.9 and b2=0.999. Along with it, we have also used ReduceLROnPlateau to reduce the learning rate so that our model shows improvement. Then, we trained the model on training dataset.

CNN Model

Our model is sequential since we are initializing this deep learning model as a sequence of layers. The information is propagated from the input layer to the hidden layer to the output layer through the model. This network employs a mathematical operation called convolution. The primary purpose of convolution is to find features in our image using a feature detector and put them into a feature map and having them in a feature map, and it preserves the spatial relationship between pixels, which is very important for us.

The first step towards building a model for our work is to initialize the model. Since our model is sequential, we initialized it as sequential. The next step is to create our first layer of CNN to perform convolution operation. We created the input layer of our CNN. We have taken 32 feature detectors of 3x3. The layer takes input into a batch of images. In our case, the batch size is 32. The shape of our image for the input layer is (28,28,1), where the height and width of the image are 28, and there is one channel as we grayscaled our images. Each feature detector produces a feature map after convolution operation over the image with an output volume of 26x26x1. We have taken the default value of padding that is 0 and stride as 1. After the convolution operation, the ReLU activation function (Figure 6.) is applied to increase non-linearity in our CNN model. If the value is greater than 0, it passes the input; otherwise, it returns 0.

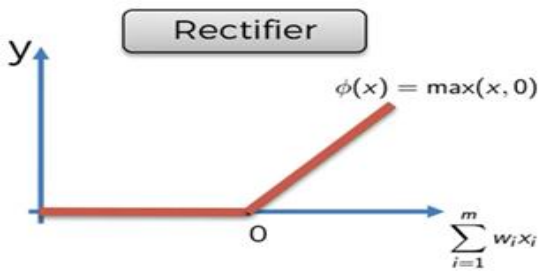

Fig. 7. ReLU Activation Function

Now we have performed Batch Normalization to keep the output of the layer of the same range to get an unbiased result. After this, the output of our first convolution layer of volume 26x26x32 is passed for the max-pooling operation to the next layer. Max pooling help to get rid of unnecessary features and account for their spatial or textual or any kind of distortion. Pooling also helps in preventing any kind of overfitting as it avoids few parameters while performing a pooling operation to get a pooled matrix. Max pooling is used to focus only on the relevant information from the image by extracting only higher values from a portion of input by using an empty feature detector. Pooling is also called downsampling. We are taking stride as two and filter feature detector size as 2x2. As a result, the height and width are reduced by a factor of 2. Thus, the output volume will be 13x13x32. Now we have a second layer, which consists of 1 convolution 2d layer and one max pooling. The input of this layer is the output of the previous layer, which is 13x13x32. The kernel size is 3x3, with 64 different feature detectors. So, the output volume is 11x11x64. Now we again applied Batch Normalization and passed the output volume 11x11x64 as input for Max Pooling, and we get the output as 5x5x64. Now we have the last layer, which is the same as all previous layers consisting of 1 conv2d layer and one max pooling layer [6]. So, the operations performed will be similar, but in the last layer, the number of feature detectors we took is 128. So, the output of the final layer will be 1x1x128[8]. Now we will perform a flattening operation to make it suitable for ANN's input. So, after flattening, it will be converted into a 128x1 vector, and it will be fed to the dense layer (Hidden layer in ANN) with some initialized weights and bias. The output of the dense layer is passed through the ReLU activation function to increase the non-linearity in the model. Now we have used a dropout layer with a dropping rate of 25%, which means 25% of random neurons are switched off so that we can prevent overfitting of our model[12]. Now the output of this hidden or dense layer is fed to the final or output dense layer with 25 neurons. This layer has 25 neurons representing 25 classes. The activation function here we have used is the Softmax activation function. Here we could have used the sigmoid function as the sigmoid function is the heart of the probabilistic approach, but the sigmoid function is good only when we classify between 2 classes. So, when there are more classes, softmax is used, and softmax ensures that the sum of the probability of the output is equal to one, which makes it easy to understand.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Fig. 8 An equation to calculate Softmax Function.

Softmax converts the linear input data into a probability array of all 25 classes, and the probabilities are checked with the actual output, and the loss is calculated using sparse categorical cross-entropy[7]. Here we have used sparse categorical cross-entropy because we have labelled as targets. Finally, we compile our model, and to minimize our cost function, we use 'adam' optimizer (stochastic Gradient Descent Method) to attain a global minimum. 'adam' optimizer is used with a learning rate of 0.001 and values of b1=0.9 and b2=0.999. During the backpropagation in the model, weights in the dense layer, as well as weights present in the feature detector changes using the local gradient if not useful for the model. The total trainable parameters to be calculated are 171,993, as seen in Figure 9., representing each layer with its output shape and trainable parameters.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 26, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
dropout (Dropout)            (None, 11, 11, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 11, 11, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 3, 128)         512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 25)                12825
=================================================================
Total params: 172,441
Trainable params: 171,993
Non-trainable params: 448
```

Fig. 9. Trainable Parameters

C) Independent and Dependent Variables

Independent variable- number of images which is 27455, pixel distribution of images which is stored in NumPy array[10].
Dependent variables are- feature map after each convolution layer output, the volume of feature map which is input to the next layer, probabilities of labels which is used for classification.

D) Parameters and Hyper Parameters

Parameters like kernel size, astride of filter, padding, number of features/filters, filter size, activation function, a connection between two layers, pooling type, number of labels[11].
Hyperparameters like coefficients in the learning rate, dropout rate, batch size, number of the hidden and dense layer in architecture, number of epochs.

E) Performance Measure

Performance measures are used to check the correctness of our model. The majors we are going to use are Accuracy on both training and validation dataset, loss in both validation and training dataset, Precision, Recall, and F1 measure of all 25 classes.

F) Validation Method

We are using the Holdout validation technique to evaluate our model. We initially have a training and testing set. Our dataset contains 34,627 images. 20% of total images (7172) are present in our testing set and the rest in the training set[9]. We split our training set into two parts, one for training our model and another part which is a validation set for validating our model[12-14]. Later we will analyze our model based on predictions predicted by our trained model on the test set and obtain its accuracy, precision, recall, F1 Score and AUC Score to see its performance.

IV. RESULT AND ANALYSIS

ANN Model :
After training the model , it was tested on testing dataset. We have used performance measures like accuracy, precision, recall and f1 score. The results are shown in table1 :

	Accuracy	Precision	Recall	AUC Score	F1 Score
Result	0.8353	0.83	0.82	0.9142	0.82

Table 1. Accuracy, Precision, Recall, AUC Score, and F1 Score obtained from the test

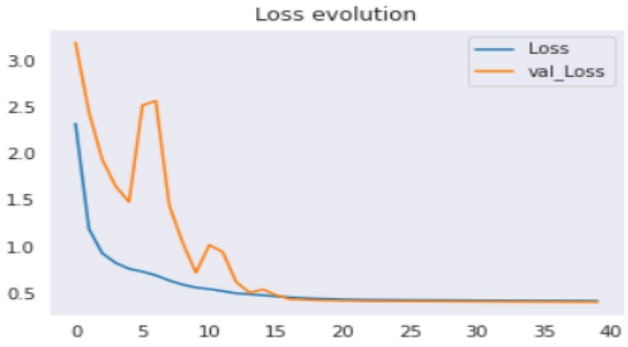

Fig.10. Loss Evolution

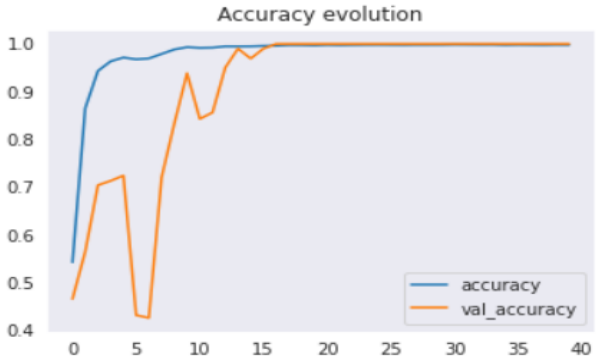

Fig . 11 Accuracy Evolution

The classification report for ANN model is as follows:

	precision	recall	f1-score	support
0.0	0.84	1.00	0.91	331
1.0	0.97	0.97	0.97	432
2.0	0.94	0.98	0.96	310
3.0	0.95	1.00	0.98	245
4.0	0.91	0.96	0.93	498
5.0	0.79	0.92	0.85	247
6.0	0.90	0.84	0.87	348
7.0	0.99	0.91	0.95	436
8.0	0.94	0.78	0.85	288
10.0	0.88	0.63	0.73	331
11.0	0.89	1.00	0.94	209
12.0	0.82	0.86	0.84	394
13.0	0.86	0.53	0.66	291
14.0	1.00	0.83	0.91	246
15.0	0.94	1.00	0.97	347
16.0	0.75	1.00	0.85	164
17.0	0.21	0.43	0.28	144
18.0	0.76	0.68	0.72	246
19.0	0.60	0.60	0.60	248
20.0	0.60	0.69	0.64	266
21.0	0.93	0.73	0.82	346
22.0	0.86	0.64	0.74	206
23.0	0.81	1.00	0.90	267
24.0	0.77	0.71	0.74	332
accuracy			0.84	7172
macro avg	0.83	0.82	0.82	7172
weighted avg	0.85	0.84	0.84	7172

Accuracy: 83.53318460680424

Fig . 12 Classification Report

CNN Model :

Once the network was trained, it was tested on the testing dataset, including 7,712 samples that were not seen by the network at the time of training, After running it against the testing dataset, performance measures like accuracy, precision, recall, AUC score, and f1 scores were measured, Table 2 summarizes the results.

We trained our model on 20 epochs with a batch size of 512. The maximum accuracy obtained on the training set was 99.60%, and invalidating set was 99.95%—epochs. The accuracy graph is shown in Figure 13 and Figure 14 for training and validation accuracy, training and validation loss, respectively.

	Accuracy	Precision	Recall	AUC Score	F1 Score
Result	0.9963	0.99	1.00	0.9981	1.00

Table 2. Accuracy, Precision, Recall, AUC Score, and F1 Score obtained from the test

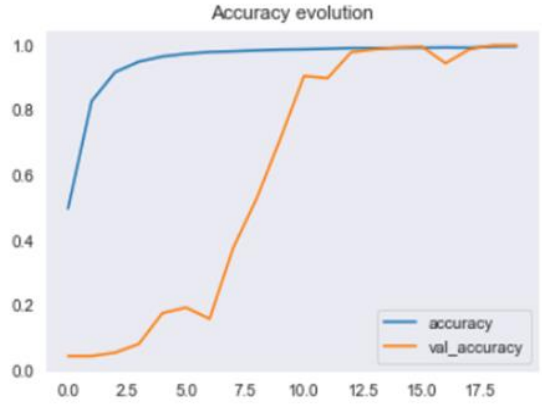

Fig. 13 Accuracy evolution

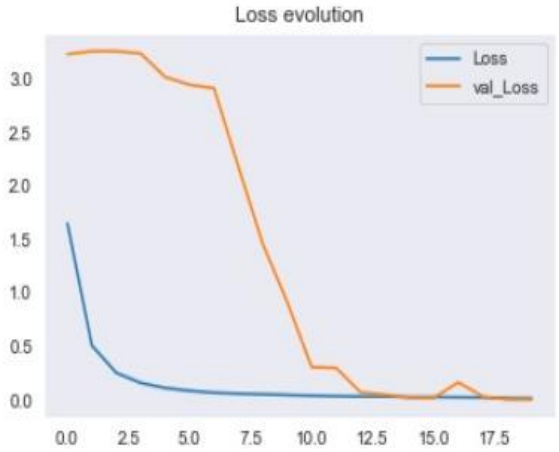

Fig. 14 loss evolution

Along with the calculated performance measures as shown in table 2, we also obtained a confusion matrix. A confusion matrix gives a depiction of the true positives and negatives obtained after running the model. This gives a better understanding as to where the model was faulty and how many true negatives or false positives are there.

The confusion matrix helps in deducing that most of the predictions made by the model are correct. However, some errors can be suspected due to the similarity in some letters. For instance, the fingers' positioning is very similar for O and S, which leads to some faulty predictions. Since we are using the MNIST dataset, most of the predictions are correct. Some of the predictions of our network are shown in Figure 15.

Fig. 15. Predictions of the network on the test set.

The Classification report obtained at the end is shown in Figure 16. And precision, Recall, F1 Score for each label is also shown.

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	331
1.0	1.00	1.00	1.00	432
2.0	1.00	1.00	1.00	310
3.0	1.00	1.00	1.00	245
4.0	1.00	1.00	1.00	498
5.0	0.99	1.00	1.00	247
6.0	1.00	0.99	0.99	348
7.0	0.99	1.00	1.00	436
8.0	1.00	1.00	1.00	288
10.0	1.00	1.00	1.00	331
11.0	1.00	1.00	1.00	209
12.0	1.00	1.00	1.00	394
13.0	1.00	1.00	1.00	291
14.0	1.00	0.99	1.00	246
15.0	1.00	1.00	1.00	347
16.0	1.00	1.00	1.00	164
17.0	0.93	1.00	0.96	144
18.0	1.00	1.00	1.00	246
19.0	1.00	0.97	0.99	248
20.0	1.00	1.00	1.00	266
21.0	1.00	0.97	0.98	346
22.0	1.00	1.00	1.00	206
23.0	0.97	1.00	0.99	267
24.0	1.00	1.00	1.00	332
accuracy			1.00	7172
macro avg	0.99	1.00	1.00	7172
weighted avg	1.00	1.00	1.00	7172

Accuracy: 99.63747908533185

Fig. 16. Classification Report

V. CONCLUSION AND FUTURE WORK

This paper exhibits a method based on artificial neural networks and convolutional neural networks for recognizing the sign language alphabet. ANN model achieved an accuracy of 83.53%. The accuracy obtained is low as the model overfitted. We used various techniques to remove overfitting in our model which include Batch Normalization, dropout layers, Regularization, and early stopping. Despite of applying all these techniques, the ANN model did undergo overfitting and that's why we decided to opt CNN to train our model. The proposed CNN Model was able to achieve an accuracy of 99.63% on our test dataset. Initially, when we were using only one convolutional layer there was overfitting in the dataset and to overcome this issue of overfitting, we added two more layers so that we can drop some parameters and still preserve the essential features of images. Moreover, we also used the dropout layer, Batch Normalization, and data augmentation to overcome the same and achieve a better performance of our network.

The letters like 'J' and 'Z' were not included in the classification due to the hand movement. So, video frames are required to classify them. The study may be carried out in our work to accept video frames and classify letters like 'J' and 'Z' as well. Moreover, a large dataset of Sign Language is required as the available dataset is quite constrained and most of the researchers are carried out on the same making it more difficult to compare the work of different researchers.

ACKNOWLEDGMENT

The authors are thankful to the central library of Delhi Technological University Delhi, India for providing all the materials and references required for the lab and research work.

REFERENCES

[1] S. Y. Kim, H. G. Han, J. W. Kim, S. Lee, and T. W. Kim, "A hand gesture recognition sensor using reflected impulses," *IEEE Sens. J.*, vol. 17, no. 10, pp. 2975–2976, 2017, doi: 10.1109/JSEN.2017.2679220.

[2] Y. Cui and J. Weng, "A learning-based prediction-and-verification segmentation scheme for hand sign image sequence," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 8, pp. 798–804, 1999, doi: 10.1109/34.784311.

[3] A. Kumar and A. Kumar, "Dog Breed Classifier for Facial Recognition using Convolutional Neural Networks," pp. 508–513, 2020.

[4] K. L. Bouman, G. Abdollahian, M. Boutin, and E. J. Delp, "A low complexity sign detection and text localization method for mobile applications," *IEEE Trans. Multimed.*, vol. 13, no. 5, pp. 922–934, 2011, doi: 10.1109/TMM.2011.2154317.

[5] J. Wu, L. Sun, and R. Jafari, "A Wearable System for Recognizing American Sign Language in Real-Time Using IMU and Surface EMG Sensors," *IEEE J. Biomed. Heal. Informatics*, vol. 20, no. 5, pp. 1281–1290, 2016, doi: 10.1109/JBHI.2016.2598302.

[6] C. Zhang, W. Ding, G. Peng, F. Fu, and W. Wang, "Street View Text Recognition With Deep Learning for Urban Scene Understanding in Intelligent Transportation Systems," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–17, 2020, doi: 10.1109/tits.2020.3017632.

[7] M. Safeel, T. Sukumar, K. S. Shashank, M. D. Arman, R. Shashidhar, and S. B. Puneeth, "Sign Language Recognition Techniques- A Review," *2020 IEEE Int. Conf. Innov. Technol. INOCON 2020*, pp. 1–9, 2020, doi: 10.1109/INOCON50539.2020.9298376.

[8] W. Aly, S. Aly, and S. Almotairi, "User-independent american sign language alphabet recognition based on depth image and PCANet features," *IEEE Access*, vol. 7, pp. 123138–123150, 2019, doi: 10.1109/ACCESS.2019.2938829.

[9] S. Hayani, M. Benaddy, O. El Meslouhi, and M. Kardouchi, "Arab Sign language Recognition with Convolutional Neural Networks," *Proc. 2019 Int. Conf. Comput. Sci. Renew. Energies, ICCSRE 2019*, pp. 2019–2022, 2019, doi: 10.1109/ICCSRE.2019.8807586.

[10] Y. Li, X. Wang, W. Liu, and B. Feng, "Pose Anchor: A Single-Stage Hand Keypoint Detection Network," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 7, pp. 2104–2113, 2020, doi: 10.1109/TCSVT.2019.2912620.

[11] X. Shen and F. L. Chung, "Deep network embedding for graph representation learning in signed networks," *arXiv*, vol. 50, no. 4, pp. 1556–1568,

2019.
[12] Y. Zhu, M. Liao, M. Yang, and W. Liu, “Cascaded Segmentation-Detection Networks for Text-Based Traffic Sign Detection,” *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 209–219, 2018, doi: 10.1109/TITS.2017.2768827.