Dr. Jalaja S, et. al. International Journal of Engineering Research and Applications
www.ijera.com
ISSN: 2248-9622, Vol. 12, Issue 10, October 2022, pp. 92-99

RESEARCH ARTICLE OPEN ACCESS

Robotic Arm for Sign Language Interpretation with sentiment analysis and auto-complete text features

Dr. Jalaja S
Department of Electronics and Communication, Bangalore Institute of Technology, KR Road, Bengaluru

Kiruthiga Chandra Shekar
Department of Electronics and Communication, Bangalore Institute of Technology, KR Road, Bengaluru

ABSTRACT
This paper presents the design of a low-cost, portable robotic arm for American Sign Language (ASL) interpretation and sentiment analysis. To mitigate the complexity of communication between the deaf and dumb and common beings, a two-way communication and interpretation model has been designed to translate the English language into ASL symbols on the robotic arm and ASL into the English language using image recognition. Along with translation, the model can also analyse the sentiments of the mute people and has an auto-text completion feature for ease of typing, thus improving the entire calibre of interaction and making it more human-like.

Keywords- American Sign Language (ASL); sentiment analysis; two-way communication;

Date of Submission: 07-10-2022 Date of Acceptance: 18-10-2022

I. Introduction

Impairment in hearing and speech is one of the most common disabilities worldwide. Deafness and dumbness is either congenital or acquired. Congenital deafness leads to deprivation of hearing by birth, arising due to natural causes. Acquired deafness arises from some disease, accident or other cases. Deafness is of two types- totally deaf and the partially deaf. As per WHO reports, in India, almost 63 million people are suffering from hearing impairment, majority consisting of young children between ages 0 to 14.

Deaf people communicate in two ways- sign language and lip reading. Due to a lot of similar sounding words in the English language, only 30% of the words in the language can be accurately differentiated, even by the most experienced lip reader. Sign language is a more accurate way for deaf people to communicate because they can differentiate words without communication. American Sign Language (ASL) is a purely visual language, it cannot be written, but can be translated.

Modern technology proposes hearing aids to elevate communication difficulties, but when a person loses their hearing ability entirely, hearing aids are unavailable. Behind-the-ear hearing aids are also only helpful with profound hearing loss patients. These devices are therefore, not widely popular among the deaf and dumb people and require improvement in design for various settings. In this article, we propose a design of a robotic sign language interpreter for the mute people. The unique features of the proposed design are:

1) Low-cost, 3D printed and portable robotic arm for translation of the English language into ASL symbol.
2) An image-recognition model to translate ASL into the English language, thereby facilitating two-way communication.
3) The translating model comes with sentiment analysis and text auto-completion feature for faster typing in the user interface.

The proposed setup in its current form can translate, ASL letters into English language and display on the monitor, and another person who does not know ASL can type in the monitor and the robotic arm will show the corresponding ASL symbol. In addition, the setup can analyze sentiments of the mute people and display on the monitor. This article discusses the design and construction of the proposed system and is organized as follows.

II. EXISTING METHODS

Paper [1] represents an on-screen sign language translation system using java NetBeans. Due to its high accuracy and lesser time consumption, canny edge detection algorithm has

been used. This algorithm has better noise reduction and clear image detection for future processing with lower error-rate and localized edge points. Paper [2] presents a virtual sign language translator based on gesture recognition. The input through camera is recorded, segmented into frames, and the recognized symbol is translated into corresponding English alphabet. The image fragments are passed through multiple filters for enhancement and elimination of unwanted background noise. After pre-processing, Fourier description technique is used to recognize the symbols and store in the respective directory. Paper [3] exposes a neural network based gesture recognition system, that has an accuracy of 100%. Only a few gestures have been trained and tested using Support Vector Machine machine learning technique. Paper [4] reveals a real-time image recognition model using Haar features with AdaBoost classifier to differentiate between the right and left arm and various skin colors. Paper [5] shows a sign language interpreter using SciPy to perform perceptron technique in neural networks. Paper [6] represents the use of ARM CORTEX A8 processor for implementation and OpenCV for image recognition in real-time. Paper [7] presents an Indian Sign Language numbers recognition system, trained with dataset containing 300 images and has a prediction accuracy of 99.56% in 22 epochs. Paper [8] presents a system that makes use of convolutional Neural network for spatial features extraction and inception model for sign language recognition. Long Short Term Memory combined with Recurrent Neural network is used to extract temporal features from the input video.

III. PROPOSED METHOD

The paper proposes an efficient, portable, low-cost model for the two-way communication system. Currently, the design is composed of three Machine learning image recognition models for recognizing ASL letters shown to the webcam and displaying corresponding English language alphabets on the user-interface, which in this case is the laptop screen. This completes one-way communication between the mute person and the common person.

The other way of communication is established when a person, who does not know ASL wants to convey a message to a deaf and dumb person, he/she can type the English alphabet on the keyboard and a 3D printed robotic arm will show the corresponding ASL symbol for the letter as shown in figure 1.

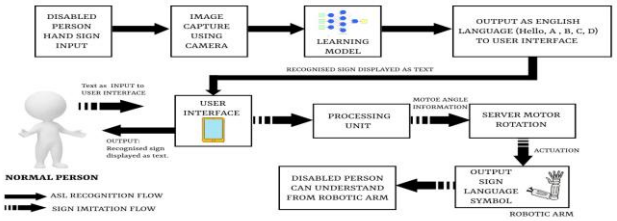

Fig.1. Block diagram of ASL interpreter

IV. METHODOLOGY

Implementation of image-recognition model:

The first step is to import necessary packages for building the model. The packages include pandas, numpy, seaborn, matplotlib, tensorflow, keras and sklearn. Once the basic packages have been imported, deep learning packages are imported after tensorflow gpu has been installed using .whl file. tf.test.is_built_with_cuda() and tf.test.is_built_with_gpu_support() commands are used to test if cuda and gpu support have been enabled. Another os module is imported to explore the dataset used for training the deep learning model. Os.listdir(train_dir) is used for checking contents of train and test directory. Plt.imshow(a1) command is used to check one image in directory to confirm if the correct directory has been accessed. For each image in the dataset, average size of the images is calculated. For loops are used to go through each image in training and testing directory and normalizing them using pillow. This step is optional since we can use OpenCV and in experimentation we have higher accuracy. Hence, the for loops have been commented. Next, one image is accessed from each class like A, B, C, D, del, E, F, G, H, I, J, K, L, M, N, nothing, O, P, Q, R, S, space, T, U, V, W, X, Y, Z, bearing unique labels are in the right directories.

The next important phase of the image recognition model is training the dataset. A tensorboard variable is setup to check the training epochs of deep learning models. A function is created to access all images across all twenty-nine classes and a training and test split is created using scikit-learn. Using the ImageDataGenerator() function, an image data generator is created. Using the image_gen.random_transform(), image augmentation is added to the existing dataset. Next, a model has to be defined for image recognition, once the dataset has been cleaned and prepared. The flow diagram and detailed explanation of the model is shown in figure. Image augmentation introduced to the image dataset include the rotation, horizontal flip, vertical flip, addition of sheer, padding and other such noise inducing

operations which may introduce variety into the dataset and prevent over fitting on the image data and ensure a good validation accuracy for the trained model. In addition to pillow, openCV can be used to carry out important image processing operations such as background elimination and sckit-learn can be used to replace image data generators by invoking the train test split functionality, which would improve the efficiency.

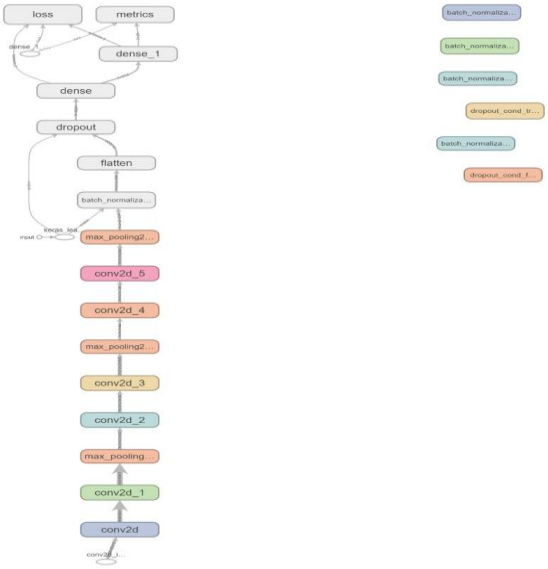

Fig.2. Flowchart of image recognition deep learning model

The first step is to create Convolutional Neural Network of suitable architecture. The flowchart for the image recognition model is shown in figure 2. The architecture is defined as follows:
The Sequential model is initialized with the following layers:
1 X Conv2D with 16 filters.
1 X Conv2D with 32 filters.
1 X MaxPool2D layer
1 X Conv2D with 32 filters.
1 X Conv2D with 64 filters.
1 X MaxPool2D layer.
1 X Conv2D layer with 128 filters
1 X Conv2D layer with 128 filters
1 X MaxPool2D layer
1 X BatchNormalization
1 X Flatten layer
1 X Densely Packed layer with 52 perceptrons with activation function = 'relu'(rectified linear units)
1 X Densely Packed layer with 29 neurons(number of classification) with "Softmax" Activation function.
Optimizer chosen is 'adam','rmsprop' is a suitable alternative.
Loss function is 'categorical cross entropy'.
Kernel regularizer is 'l2'.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 16)	448
conv2d_1 (Conv2D)	(None, 64, 64, 32)	4640
max_pooling2d (MaxPooling2D)	(None, 21, 21, 32)	0
conv2d_2 (Conv2D)	(None, 21, 21, 32)	9248
conv2d_3 (Conv2D)	(None, 21, 21, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 7, 7, 64)	0
conv2d_4 (Conv2D)	(None, 7, 7, 128)	73856
conv2d_5 (Conv2D)	(None, 7, 7, 256)	295168
max_pooling2d_2 (MaxPooling2	(None, 2, 2, 256)	0
batch_normalization (BatchNo	(None, 2, 2, 256)	1024
flatten (Flatten)	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense (Dense)	(None, 512)	524800
dense_1 (Dense)	(None, 29)	14877

Total params: 942,557
Trainable params: 942,045
Non-trainable params: 512

Fig.3. Model Summary

An Early stopping variable is created to stop model training of suitable accuracy and prevent over fitting. The model is compiled and the training data is fit. To check the model performance, a model performance metrics are plotted. The test data is then loaded. The test data contains images with the image name as the class, to which the image belongs to. In this way, the model is tested for accuracy through several trials.
Implementation of image-recognition model using DenseNet
Import necessary packages such as pandas, matplotlib, seaborn, numpy, cv2, sklearn, Tensorflow and its sub packages such as preprocessing, models, layers, optimizers, callbacks and applications. Use the Os module to explore the dataset used for training the DL module.

The next step is taking one image in each directory to see if the correct directory is chosen. Plt.imshow(a1) command is used to check one image in directory to confirm if the correct directory has been accessed. For each image in the dataset, average size of the images is calculated. For loops are used to go through each image in training and testing directory and normalizing them using pillow. This step is optional since we can use OpenCV and in experimentation we have higher accuracy. Hence, the for loops have been commented. Next, one image is accessed from each class like A, B, C, D, del, E, F, G, H, I, J, K, L, M, N, nothing, O, P, Q, R, S, space, T, U, V, W, X, Y, Z, bearing unique labels are in the right directories.

To monitor the real time training of the DenseNet model, tensorboard variable can be setup which also helps in the illustration of the flow chart. We create a function to access images across all 29 classes and using scikitlearn framework, training, testing and validation datasets have been created.

To make sure that the DenseNet model is suitable for real time applications, noise is introduced into the data set by using the image data generator to add rotation, vertical and horizontal flip, sheer and padding to every third image in the dataset.

The model architecture is illustrated in figure 4 as follows:

```
model_d=DenseNet121(weights='imagenet',include_top=False, input_shape=(64, 64, 3))
x = model_d.output
x = GlobalAveragePooling2D()(x) x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(1024,activation='relu')(x)
x = Dense(512,activation='relu')(x) x = BatchNormalization()(x)
x=Dropout(0.5)(x)
preds=Dense(29,activation='softmax')(x) #FC-layer
```

Before trainable parameter optimization:
Total parameters: 8,632,925
Trainable parameters: 8,546,205
Non-trainable-parameters:86,720

```
model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

After trainable parameter optimization:
Total parameters: 8,632,925
Trainable parameters: 1,592,349
Non-trainable parameters: 7,040,576

The next step is compiling the model and fitting the training data. The model is trained for total of 50 epochs. The number of epochs was chosen based on the grid search approach.

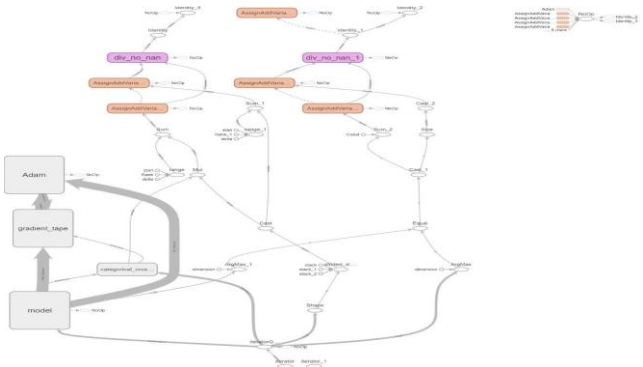

Fig.4. Flowchart of image recognition deep learning model using DenseNet

Implementation of sentiment analysis model

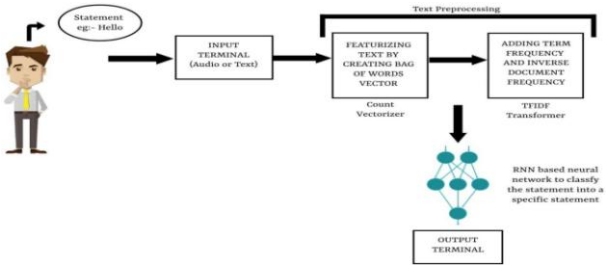

Fig.5. Block Diagram of sentiment analysis model

Figure 5 represents the block diagram of sentiment analysis model, where, a sentence is taken as input either in voice or text format and the input text is featurized by creating bag of words vector and further processing is done to determine term frequency and inverse document frequencies. This completes the text processing part of sentiment analysis. After the pre-processing, the words are passed through the RNN based neural network to classify the statement into specific sentiments. The detailed process is described below.

Data acquisition-The Data required to train a sentiment recognizing neural network can be created manually, individual records in the training data would consist of the example statement and a label representing the sentiment of the statement.

Data Cleaning, Data Visualization and Exploratory Data Analysis- The data can be compiled into a data structure such as a dataframe by using the pandas library, matplotlib library can now be utilized to observe the trends in data such as length of the statement associated with each sentiment. In data cleaning the primary goal is ensure that data has balanced class representation and minimize noise in data which will be utilized to train the neural network. A heatmap, as shown in figure 6, is used to check if there is any noise in the data. The heatmap below represents that there are no null values or any noise in the data.

Fig.6. Exploratory data analysis heatmap

Text Preprocessing- the nltk package can be imported, the download shell of nltk provides a wide variety of sub packages such as "stop words" which helps us featurize the textual data and

eliminate the words in a sentence which have no impact on emotion.eg: "Hello how are you?" can be after being filtered for stop words will be transformed into [“Hello”].Punctuations can also be eliminated in the preprocessing of text, the string package can be utilized to filter out the punctuations in a statement. Count Vectorizer-The list of words after text preprocessing can now be featured by being converted into a bag of words vector. Pre-processing functions are used to convert each message, represented as a list of tokens (lemmas), into a vector that machine learning models can understand. The process follows these steps: count how many times does a word occur in each message (Known as term frequency), weigh the counts, so that frequent tokens get lower weight (inverse document frequency), normalize the vectors to unit length, to abstract from the original text length (L2 norm). Each vector will have as many dimensions as there are unique words in the data frame. First SciKit Learn's CountVectorizer has been used. This model will convert a collection of text documents to a matrix of token counts. This can be imagined as a 2-Dimensional matrix, as shown in figure 7, where the 1-dimension is the entire vocabulary (1 row per word) and the other dimension are the actual documents.

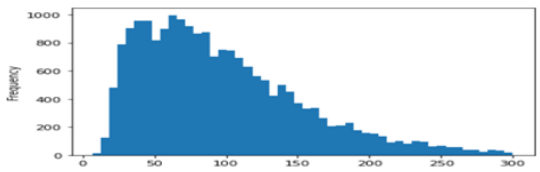

Fig.7. Frequency of occurrence of words

TF-IDF Transformer-TF-IDF stands for term frequency-inverse document frequency, and the tf-idf weight is a weight used for retrieving information and mining text. The weight is a statistical measure to evaluate the importance of word in a collection. The importance of the word is determined by the number of times of occurrence in the document and offset by its frequency in the collection.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency is a measure of the importance of a term. When Term frequency is computed, all terms hold equal importance, whereas, certain words like “is”, “that”, “of”, “this”, may occur too many times in a document but have very little importance in a document. Therefore, such frequent terms must be weighed down and the rare words must be scaled up, using inverse document frequency.

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

TF(t) and IDF(t) will serve to augment the vector, which will be fed into a neural network. After

```
In [40]: model.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (128, None, 64)           5632

 gru (GRU)                   (128, None, 1028)         3373896

 dense (Dense)               (128, None, 88)           90552

=================================================================
Total params: 3,470,080
Trainable params: 3,470,080
Non-trainable params: 0
_________________________________________________________________
```

usage of Countvectorizer and TfIdf. Transformer the output will be a sparse matrix, which can now be feed into a neural network of suitable architecture. The choice of the type of neural architecture is dependant on the required application, RNN built using LSTMs and GRU’s are a suitable option, but a much more efficient approach would be to use the scikit learn’s multinomial naïve bayes classifier, since its prediction is based on a ensemble of most probable hypothesis rather than considering a single most probable hypothesis, sckit learn does include other Gaussian classifiers which could also be implemented to obtain required functionality and serve as a multiclass/binary classifier.

Implementation of Autocomplete text model

The first step is to setup custom environment with cuda toolkit enabled and required versions of necessary packages such as pandas, numpy, seaborn, matplotlib, os, tensorflow and sklearn.

From online repositories like guttenberg.org a novel in .txt format with utf-8 encoding has been downloaded. A function is created that finds the unique characters in the novel that will be used for training the RNN, The functions returns a data structure that serves the purpose of a vocabulary, which can be used to encode the text. Two additional Data structure

-one dictionary to encode text.

-one numpy array to decode the encoded text.

The textual data can be encoded into numerical format using the data structures just created. Now, a character Dataset is created from tensor slices(encoded text)(tensor slices are a data structure that can be used to create sequences). On the character dataset .batch() function can be used to create sequential batches. Before we create a RNN model we create a custom loss function to replace "categorical crossentropy", the loss function created is based on sparse categorical

crossentropy with the parameter from_logits set to True.

We now create a RNN model of the following architecture.

The flowchart of the RNN architecture used is described below. The model consists of embeddings layer to map the input sequence into a numerical array of dimension equal to the vocabulary.The next layer consists of GRU (gated recurrent units)-output of the GRU takes into consideration not just the present input but history for previous outputs of the perceptron GRU perceptron.Long short term memory units(LSTMs can also be used) Finally, we have a dense fully connected layer with number of units equal to number of possible classification. The custom loss function is invoked and adam is used as the optimizer.

The model is trained for a suitable number of epochs, which can be determined using the Grid search approach. The saved model can be saved, the weights of the saved model can be loaded onto a newly created model with a different batch size if necessary. The model summary is represented in figure 8.

```
In [40]: model.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (128, None, 64)           5632

 gru (GRU)                   (128, None, 1028)         3373896

 dense (Dense)               (128, None, 88)           90552

=================================================================
Total params: 3,470,080
Trainable params: 3,470,080
Non-trainable params: 0
_________________________________________________________________
```

Fig.8 Model summary

Implementation of translation of English letters to ASL using Robotic arm

Arduino UNO microcontroller has been used to control the rotation of the servo motors, which in turn help rotate the fingers of the 3D printed robotic arm, to represent the corresponding ASL letter. Each micro servo motor requires 5V of power supply for operation and the robotic arm contains 5 servo motors. Arduino IDE has been used to code the instructions for the rotation of the micro servos. To supply 5V of power to each servo motor, several steps are involved as shown in figure 9. The first step is to step down the 220V AC main power supply to 12V AC using a 220V-12V step down transformer. Once, the main power supply has been stepped down, a bridge rectifier has been used to convert AC to 12V DC power. Next, L7805CV, a three terminal a linear voltage regulator IC is used to regulate the 12V DC input to exactly 5V. Thus, the required power for servo motors has been setup.

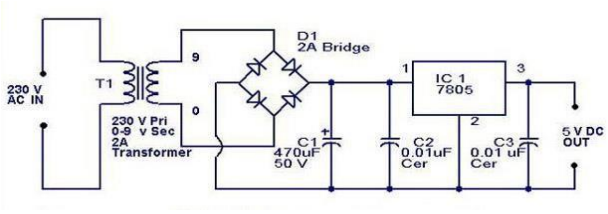

Fig.9. Circuit diagram for powering Robotic arm

Now, each servo motor connected to each finger needs to be given instructions of open and close when a certain letters on the keyboard is pressed. In this project, a left humanoid robotic arm has been used, hence, only left hand ASL letters can be translated. These letters include A, B, E, F, K, W, I, D and L. For each letter, an Arduino code has been written to rotate each upto 180, 270 or 360 degrees, depending on the ASL symbol.

V. RESULTS

The accuracy and loss plots for each deep learning model have been shown in the figures below. Figure 10 represents the accuracy and loss plots for image recognition model. The image recognition model has a prediction accuracy of 98.46% with an evaluating loss of 0.151217. Similarly, Figure 11 shows the image recognition model using DenseNet architecture and image augmentation. The image-recognition model using DenseNet architecture and image augmentation has a prediction accuracy of 98.07% with evaluation loss of 0.062042. Figure 12 presents the recognition of ASL symbol and displaying its English Language equivalent.

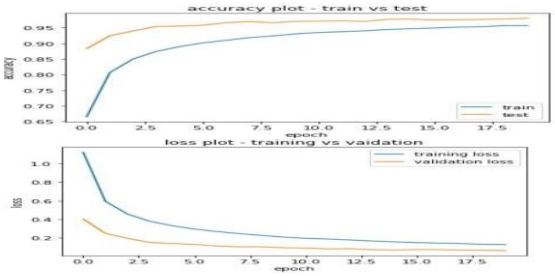

Fig.10. Accuracy and loss for plot image-recognition model

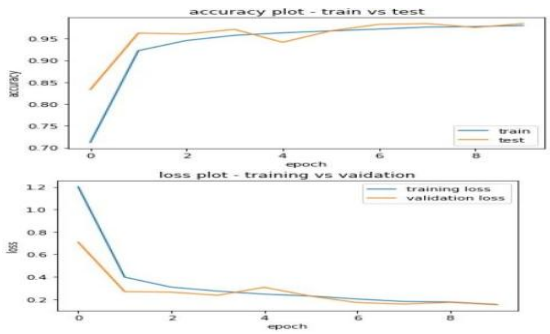

Fig.11. Accuracy and loss for plot image-recognition model using DenseNet architecture and image augmentation

```
size_new = 64,64
temp_new = cv2.imread("C:\\Users\\ASUS\\Downloads\\archive\\asl_alphabet_test\\A\\A2014.jpg")
temp_new = cv2.resize(temp_new, size_new)
image_new = np.array(temp_new)
image_new = image_new.astype('float32')/255.0
image_inp = np.expand_dims(image_new,axis=0)
answer=model.predict(image_inp).argmax()
prediction = processed_label[answer]
plt.imshow(image_new)
print(f"The image is indicating {prediction}")
```

```
The image is indicating A
```

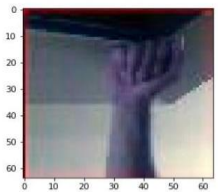

Fig.12. Prediction of ASL symbol 'A' shown

Table 1 test accuracy obtained for various architectures

Architecture	Before Pruning		After Pruning	
	Test Accuracy	Memory size	Test Accuracy	Memory size
DenseNet-201	88.6%	98.27 Mb	84.8%	61.43 Mb
DarkNet-53	86.4%	70.30 Mb	82.1%	42.78 Mb
ResNet-50	86.5%	76.65 Mb	82.4%	52.85 Mb

From paper [9], the test accuracy of the DenseNet architecture model used for recognizing Bangladesh Sign Language is 88.6%, as shown in table 1. The proposed model has been able to improve the efficiency of the model using the same architecture upto 98.07% to recognize American Sign Language images and convert them to English letters.

The sentiment analysis model was tested with several sentences of different emotions and showed a prediction accuracy of 93%. The classification report for sentiment analysis is shown in figure 13 along with the test results. The auto-complete text model completed the text entered for all words as shown in the figure 14. The robotic arm ASL symbol translation is shown in figure 15, where letters 'D' and 'W' are represented in ASL.

	precision	recall	f1-score	support
negative	0.91	0.97	0.94	3324
positive	0.96	0.88	0.92	2676
accuracy			0.93	6000
macro avg	0.94	0.93	0.93	6000
weighted avg	0.93	0.93	0.93	6000

Testing model by giving it sentences to classify

```
In [44]: def classify_sentence(model,text):
             textdf=pd.DataFrame(data=[text],columns=["sentence"])
             answer=model.predict(textdf["sentence"])
             return answer

In [49]: answer=classify_sentence(mnb_pipeline,"i am very impressed")
         print(answer)
         ['positive']

In [55]: answer=classify_sentence(mnb_pipeline,"i love you")
         print(answer)
         ['positive']

In [56]: answer=classify_sentence(mnb_pipeline,"i hate you")
         print(answer)
         ['negative']

In [59]: answer=classify_sentence(mnb_pipeline,"wow thats awesome")
         print(answer)
         ['positive']

In [61]: answer=classify_sentence(mnb_pipeline,"why are you so stupid")
         print(answer)
         ['negative']
```

Fig.13. Classification report and test results for sentiment analysis model

```
gen_text(final_model,"how ar",2)
how are

gen_text(final_model,"hello th",2)
hello the

gen_text(final_model,"th",1)
the

gen_text(final_model,"yo",1)
you

gen_text(final_model,"fel",2)
fell?

gen_text(final_model,"yel",2)
yell.
```

Fig.14. Results for auto-complete text

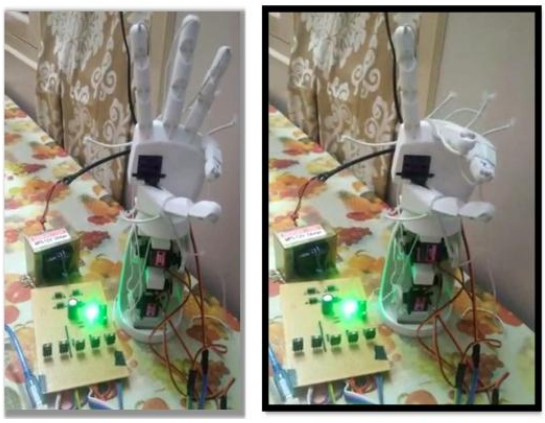

Fig.15. English to ASL letters translation by robotic arm representing D and W respectively

VI. CONCLUSION

Impairment in hearing and speech has always been a societal problem. The invention of Sign-Language has served as a mode of communication for them to be able to integrate into society but is still faced with unequal opportunities and compromises in society due to a lack of proper communication. With fast-growing technology and evolution, interference of technical agents is undeniable to help the specially challenged people to achieve equality among all kinds of people. Modern virtual interpreters and onscreen display of translated English language into Sign Language seem to have mitigated the efficacy of communication. In order to build equal opportunities for hearing and speech impaired community, an effort has been put to design a low-cost interpreter to translate ASL to English and vice versa with a relatively high accuracy and sentiment analysis and auto-complete text features. This high computational speed model can eliminate the requirement of a learned translator and can be used widely in educational and medical fields for faster communication, thus helping provide quality education and information to all people.

FUTURE SCOPE

To increase the mobility of the entire project, raspberry Pi module could be utilized i.e., the ML/DL models could be dumped onto the raspberry Pi. As an alternative to DenseNet, other modern model architectures such as EfficientNet and ResNet could be explored. The quality of dataset used for training the ML/DL models could be enhanced. Currently the project implements grid search CV for hyper parameter optimization but other heuristic search algorithms such as BFS or DFS could be used to better guarantee convergence to global minima.

REFERENCES

[1] Amitkumar Shinde, Ramesh Kagalkar," Sign Language to Text and Vice Versa Recognition using Computer Vision in Marathi," National Conference on Advances in Computing (NCAC 2015).

[2] Purva C. Badhe, Vaishali Kulkarni," Indian Sign Language Translator Using Gesture Recognition Algorithm," IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS 2015)..

[3] Rajesh Mapari, Govind Kharat ," Hand gesture recognition using Neural Network,December," International Journal of Computer Science and Network. 2012

[4] S. Wu and H. Nagahashi, "Real-time 2D hands detection and tracking for sign language recognition,"8th International Conference on System of Systems Engineering.

[5] Geethu G Nath, Anu V S," Embedded Sign Language Interpreter System for Deaf and Dumb People," International Conference on Innovations in information Embedded and Communication Systems (ICIIECS). 2017

[6] Sajanraj T D, Beena M V," Indian Sign Language Numeral Recognition Using Region of Interest Convolutional Neural Network," 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT 2018).

[7] G.Anantha Rao, K.Syamala, P.V.V.Kishore, A.S.C.S.Sastry,"Deep Convolutional Neural Networks for Sign Language Recognition"

[8] Surejya Suresh, Mithun Haridas.T.P, Supriya M.H." Sign Language Recognition System Using Deep Neural Network,"5th International Conference on Advanced Computing & Communication Systems(ICACCS) 2019.

[9] Ragib Amin Nihal,Nawara Mahmood Broti, Shamim Ahmed Deowan, Sejuti Rahman. "Design and Development of a Humanoid Robot for Sign Language Interpretation"