DOI: 10.36108/ujees/4202.60.0101
UNIOSUN Journal of Engineering and Environmental Sciences. Vol. 6 No. 1. March. 2024

DEVELOPMENT OF MOBILE APPLICATION FOR SIGN LANGUAGE TRANSLATION USING DEEP LEARNING TECHNIQUE

Omodunbi,B. A., Soladoye, A. A.*, Okomba, N. S., Asaolu, O., Ayoola, J. I., Odeyemi, P. B.

Abstract Sign Language is a visual and gestural language used by deaf and hard-of-hearing people to communicate. However, communication between hearing and non-hearing individuals can be challenging due to the language barrier. In order to overcome this barrier, a Mobile based Translator is being proposed using Convolutional Neural Network (CNN) to recognize and translate hand gestures in real-time. The proposed system consists of a CNN model trained on a large dataset of hand gestures that includes various signs, such as the alphabet, numbers, and common phrases to recognize various signs, and a backend server to handle the translation of the recognized signs into text. The system was implemented using various CNN architectures like ResNet, MobileNet and VGG-16, where the later gave the best accuracy of 97.69%. The trained VGG-16 model recognizes the signs by extracting features from the images of the hand gestures and using these features to classify the gestures. Once a gesture is recognized, the backend server translates it into text using a pre-defined mapping of signs to words or phrases. The translated text will then be displayed to the user in real-time on the mobile app, enabling seamless communication between hearing and non-hearing individuals. The proposed system was implemented as a mobile app using Flutter, which is a cross-platform development framework. This mobile app make communication easier for vulnerable and enable sharing of information without any discrimination.

Keywords: Sign language, Transfer Learning, Flutter, Gesture Recognition, Software Engineering

I. INTRODUCTION

Sign language is a visual language used by individuals who are deaf or hard of hearing to communicate with others. However, many people do not know sign language, which can lead to communication barriers for deaf individuals. Communication is a fundamental need, yet millions of people around the world face significant barriers to effective communication due to language and sensory differences [1]. Sign Language communication is expressed by movements of the hands. The most common sign language is American Sign Language (ASL), commonly used in many countries across the world and adapted for use in varying countries. The other main sign language used in Canada is la Langue des Signes Québécoise (LSQ); there is also a regional dialect, Maritimes Sign Language (MSL) [2]. Deaf and hard-of-hearing individuals, or instance, often rely on sign language as their primary mode of communication, but this can pose challenges when interacting with non-sign language users. This communication gap can lead to isolation, frustration, and missed opportunities.

To address this issue, this study developed a Mobile-based sign language translator using convolutional neural networks (CNNs) architectures and natural language processing (NLP) techniques. The goal is to provide a real-time and accurate sign language translation system that can bridge the communication gap

Omodunbi,B. A., Soladoye, A. A., Okomba, N. S., Asaolu, O., Ayoola, J. I., Odeyemi, P. B.
(Department of Computer Engineering, Federal University, Oye-Ekiti, Nigeria)
Corresponding Author: Afeez.soladoye@fuoye.edu.ng
Phone Number: 08107740087; 07047287138

between sign language users and non-sign language users. The system uses a CNN models trained on a large dataset of sign language gestures to recognize and classify different signs in real-time. This study implements the design, development and deployment of an Android & iOS based Sign-language translator that can recognize and translate American Sign Language (ASL) gestures in real-time using Convolutional Neural Network (CNN) and Natural Language Processing (NLP) engine.

Most works are on sign language recognition [3-5] which is just to detect the hand gesture and recognize it as an alphabet, number, word or sometimes sentence if it is video, this process of gesture recognition in videos is known as gloss [6]. Ordinarily, gesture recognition of sign language is not always enough in communication as the recognized gloss have to be translated into a readable language text so as to aid communication between less privileged and privileged ones. However, in recent time, researchers are now focusing on sign language translation, some of these researches are reviewed in this section so as to know the state-of-art in sign language translation.

A PC-based sign language based translator was developed by [7] using Python and TensorFlow. It translates ASL gestures to written texts with audio renderings in about one second and can match real-time gestures with equivalent gesture images at 44% similarity. The PC-based sign language translator uses machine learning for wider accessibility, however, it has limited matching accuracy. Furthermore, [8] discussed the use of image processing, specifically the Convolutional Neural Network (CNN), to convert sign language into speech or text. The authors have developed a sign detector for Indian Sign Language (ISL) to recognize numbers 1-10 and plan to extend it to recognize other gestures and expressions. This approach is widely accepted because of its potential in removing language barriers for those with hearing impairments, as widely accepted as the system is, it's had issues recognizing more complex sign language gestures and the need for further testing and refinement. Similarly, [9] proposed the need for a product that can transform sign language into a form that can be understood by common people. The authors suggest using Raspberry Pi, gesture recognition, and image detection with Python to achieve this goal. This approach makes sign language more accessible to a wider audience.

An approach for real-time recognition of Indian Sign Language using CNN and neural networks was presented by [10], with the aim of improving communication for people with hearing and speaking disabilities. The system's potential lies in its ability to provide a means of communication without the need for a translator, but is specific to Indian Sign Language. Similarly, [11] proposed a system that aims to improve the inclusion of hearing-impaired people by converting speech input into text and translating it into sign language using natural language processing and machine learning. The system is based on Python. The system has potential to increase knowledge and awareness of sign language and to facilitate communication for hearing-impaired individuals but may need further development and refinement to improve accuracy and usability. [12] Proposed an approach to improve communication between deaf and mute individuals and those without communication impairments. The approach involves training a model to recognize common sentences exchanged between buyers and sellers using hand gestures and Google Teachable Machine.

The approach uses technology to bridge communication gaps and promote inclusivity cannot account for individual variability in gestures and language use.

A sign language translator system that uses American Sign Language (ASL) dataset for recognizing the sign language alphabet and converting it into text to speech was proposed by [13]. The system achieves a 74% accuracy rate and can also be helpful for blind people. The system's hand-tracking techniques and its ability to bridge the communication gap between deaf-mute individuals and others makes it powerful. It needs improvement in accuracy and the system only recognizes ASL. [14] proposed a system for real-time American Sign Language perception using a combination of Convolutional Neural Networks and the You Only Look Once (YOLO) algorithm. The system improved communication between hearing and hearing-impaired individuals by accurately recognizing and translating sign language gestures. The system uses advanced machine learning techniques (CNN and YOLO), but may not be accurate in terms of the hand tracking and segmentation algorithms.

From all the aforementioned studies, majority of the studies employed CNN but its pre-trained architectures like VGG, ResNet and MobileNet were not explored. Also little work was done on mobile app development for sign language translation to text for easier communication between people living with disability and others that hear and speak. This is the major gap this study aimed to fill by exploring difference architectures of CNN and use the best performing architecture to develop a mobile app that can be used on smartphones for easier communication.

II. MATERIALS AND METHODS

The development of the system involves two phases, the first phase involves different stages of training and evaluating different CNN architectures and the second phase involves development of the mobile app. These stages including the deep learning technique methodology and deployment are represented in Figure 1, which shows the data acquisition, preprocessing, training and evaluation and deployment.

A. Data Acquisition

The dataset was gotten from Kaggle as used by Rahman et al., [15], the dataset composed of images of hands performing various ASL gestures or signs. But then, to ensure locality of the study and improve learning of the model and further reduce overfitting, over 2000 local dataset were additionally acquired, that featured dark skinned hands doing the sign language. These hand gestures represent all the letters and numbers in the American Sign Language. Each sample in the dataset is associated with a corresponding label indicating the ASL sign it represents. In order to localize the system, these local dataset were locally acquired by the researchers. Some of the images present in the dataset is shown in Figure 2.

The summary of the dataset used in this study is presented in Table 1 for clarification and reference. As this showed the resolution of each image present the dataset to be 64 x 64 pixel, with Jpeg extension, containing a total of 9500 images, with each class (alphabet, number, words and phrases) having 200 images and the color format is RBG. This helps in ensuring the dataset is well described for better understanding.

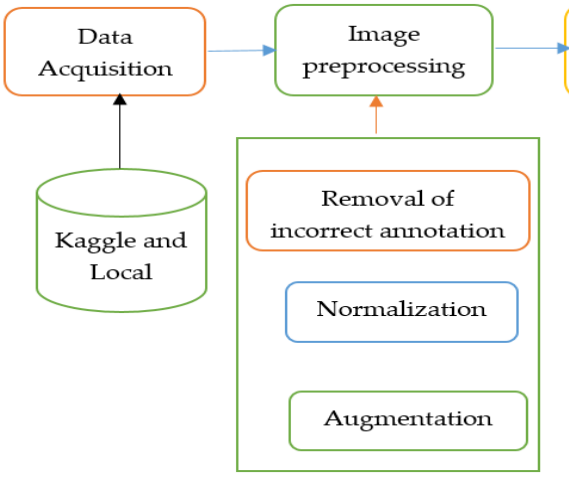

Figure 1: Overview of image recognition methodology

Figure 2: Overview of sign languages [15]

Table 1: Dataset description

Specification	Value
Resolution	64 by 64
Extension	.jpg
Number of images	9500
Number of classes	36
Number of images per class	200
File size	10kb – 20kb
Channel	3 (RGB)

B. Data pre-processing

This stage of the system development is to ensure the dataset is well process for better performance of the model. This study employed some preprocessing techniques like Data cleaning, Data augmentation

i. Data Cleaning: involves removing any irrelevant data, such as images with poor lighting or unclear hand gestures. The images with incomplete or incorrect annotations were removed. Data normalization involves scaling the data so that all the images have the same resolution, frame rate, and color space. This ensures that the model is not biased towards any particular image quality or format.
ii. Data augmentation: involves generating additional training data by applying random transformations to the existing images, such as flipping, rotating, and zooming. This helps to increase the size of the training set and improve the model's ability to generalize to new data. Techniques such as random cropping and color jittering were applied to further enhance the diversity of the training data.

C. CNN model for recognition

This study employed VGG-16 as the classification algorithm after series of experimentation have been carried on using other architectures of CNN like ResNet and MobileNet and the study found out the VGG-16 gave the best performance accuracy. VGG-16, developed by the Visual Geometry Group and Google DeepMind, is a convolutional neural network model proposed for image recognition tasks. This model has 16 layers and has a reputation for being very effective in the ImageNet Large Scale Visual Recognition Challenge. The study applied transfer learning, using the VGG16 model's learned features as a starting point for the sign language classification task. This is a common and effective strategy for deep learning, especially when the available dataset is relatively small.

This pre-trained model was modified so as to avoid and cater for domain mismatch which is one of the major drawback of employing transfer learning technique. The modification done on the original VGG-16 are highlighted as follows:

i. *Classifier_vgg16.output*: Here, the study used the output of the VGG16 model as the input to the custom layers. This output will contain the learned feature maps from the VGG16 model.
ii. *Flatten ()*: This layer is used to convert the multi-dimensional input into a single dimension array. It prepares the vector for input into the dense, feed-forward neural network layers that follow.
iii. *Dense (units=256, activation='relu'):* This is a fully connected layer where each input node is connected to each output node. The study specified 256 units, meaning there are 256 output nodes. The activation function is 'relu', or Rectified Linear Unit, which will output the input directly if it is positive, otherwise, it will output zero.
iv. *Dropout (0.6)*: Dropout is a regularization technique for reducing overfitting in neural networks. During training, 60% of the units in the previous layer will be randomly ignored. This helps to make sure the network doesn't rely too heavily on any single (or small group of) neuron(s).
v. *Dense (units=36, activation='softmax')*: This is the final layer of our network. It has 36

output nodes, which corresponds to the number of classes we have in your dataset. The 'softmax' activation function is generally used in the output layer of a network for multi-class classification problems. It gives the probabilities of the input being in each class, and all the probabilities will sum to 1.

Finally, the VGG-16 model was compiled using 'adam' as the optimizer, 'categorical_crossentropy' as the loss function (which is suitable for multi-class classification), and 'accuracy' as performance metric. The Adam optimizer is a popular choice because it combines the best properties of the AdaGrad and RMSProp optimization algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. 'Categorical_crossentropy' loss function is used as a loss function for classes greater than 2. It expects the labels to be provided in a one_hot representation i.e., For N classes, each label should be a N dimensional vector where only one element is 1 (indicating the correct class), and the rest are 0. The 'accuracy' metric computes the accuracy rate, which is the proportion of correct predictions to the target size.

The code snippet for training of the model is represented in Figure 3 which show the code for splitting the whole dataset to training and testing, and shows the output of the total number of images used for training and evaluating the model.

```
from sklearn.model_selection import train_test_split

#X_train, X_test, y_train, y_test = train_test_split(images_inv, labels, test_size = 0.2, stratify = labels)
X_train, X_, y_train, y_ = train_test_split(images, labels, test_size = 0.3, stratify = labels)
X_eval, X_test, y_eval, y_test = train_test_split(X_, y_, test_size = 0.5)

n = len(uniq_labels)
train_n = len(X_train)
test_n = len(X_test)

print("Total number of symbols: ", n)
print("Number of training images: " , train_n)
print("Number of testing images: ", test_n)

#eval_n = len(X_eval_inv)
eval_n = len(X_eval)
print("Number of evaluation images: ", eval_n)
```

Python

```
Total number of symbols:  36
Number of training images:  5040
Number of testing images:  1080
Number of evaluation images:  1080
```

```
print(y_train.shape)
```

Python

```
(5040,)
```

Figure 3: Code snippet for splitting the dataset

D. Model performance evaluation

The model was evaluated to ascertain its performance before it as integrated into the frontend for functionality. This study employed the common evaluation matric which is accuracy, this metrics is mathematically represented as given in (1)

$$Accuracy = \frac{T_p+T_n}{T_p+F_p+T_n+F_n} \quad (1)$$

E. Frontend development

The project frontend is built on Flutter. Written in the Dart Programming Language. Flutter is an open source UI framework developed by Google for creating natively compiled applications for mobile, web and desktop. It is particularly renowned for its efficiency in building high-quality, visually appealing user interfaces. Flutter utilizes a unique approach called "widget" architecture, where user interface components are represented as widgets. These widgets are customizable and can be composed together to create complex user interfaces. Flutter provides a rich set of pre-designed widgets that cover everything from basic buttons to advanced layouts, making it easier to build engaging and consistent user experiences.

To set up a Flutter project, the following steps were employed:

i. Installed Flutter Dependencies, as it has some platform-specific dependencies.

ii. Created a new Flutter project in a terminal by navigating to the directory to create the Flutter project.

iii. Ran the command 'flutter create motion_verse' to generate a new Flutter project.

iv. Used the 'cd' command to navigate into the newly created project directory 'cd motion_verse'.

v. Ran the application by connecting a physical device. In the terminal, ran 'flutter run' to compile and launch the Flutter application on the connected physical device.

vi. Itched to the 'lib/main.dart' file to start building the application.

Figure 4 shows how the trained and evaluated model was integrated into the frontend for userability. This is done using the function '*runModel*' that takes in the Camera Image parameter. The '*runModelOnFrame*' function is an in-built TensorFlow Lite function that starts by loading the pre-trained model. It is designed to perform inference on a selected image frame. The function is responsible for loading the model, pre-processing the input image, running the inference, and returning the output. This function got called from other parts of the application whenever image inference was required, reducing code duplication.

Figure 5 shows how the *'loadModel'* function was responsible for loading the pre-trained model onto the device's memory. It takes parameters of the model and label, the function ensures that necessary memory space is reserved to store the parameters and intermediate computation results. The model parameter takes in the model file that contains the model's architecture and the trained weights. The loaded interpreter and tensor details loaded on the application's initialization or when the model is required for inference

```
Future<void> runModel(CameraImage? img) async {
  if (_modelLoaded) {
    List? recognitions = await Tflite.runModelOnFrame(
      bytesList: img!.planes.map((plane) {
        return plane.bytes;
      }).toList(), // required
      imageHeight: 64,
      imageWidth: 64,
      numResults: 3,
      imageMean: 127.5,   // defaults to 127.5
      imageStd: 127.5,    // defaults to 127.5
      // rotation: 90,        // defaults to 90, Android only
      threshold: 0.1,
        asynch: true        // defaults to true

    );
    CameraService.convertYUV420ToImage(img);
    // shows recognitions on screen
    if (recognitions!.isNotEmpty) {
      print(recognitions[0].toString());
      print('Converted Imageee height: ${img.height}');

      if (_recognitionController.isClosed) {
        // restart if was closed
        _recognitionController = StreamController();
      }
      // notify to listeners
      _recognitionController.add(recognitions);
    }
  }
```

Figure 4: Code snippet for model integration

```
Future<void> loadModel() async {
  try {
   _recognitionController.add([]);
    await Tflite.loadModel(
      model: "assets/model.tflite",
      labels: "assets/label_new.txt",
    );
    _modelLoaded = true;
  } catch (e) {
    print('error loading model');
    print(e);
  }
}
```

Figure 5: Snippet of the model and label

F. Implementation and experimental setup

The study was implemented on Visual Studio Code (VS Code) - a versatile and widely-used source code editor known for its lightweight design and powerful features. Being a system built on deep learning model, Tensor Flow was employed which enabled the creation and training of deep learning models to recognize and translate sign language gestures into text or speech. ImagePro was also used, this mobile application was used to gather the local dataset. And Matplotlib is the library that provides powerful visualization capabilities that can help in understanding the data.

III. RESULTS AND DISCUSSION

This section provides the reader with the experimental results obtained while training different CNN architectures to select the best for integration into the mobile app and the interfaces of the developed sign language translation mobile app.

A. Experimental results of the CNN models

During the training of the three models, their average training and validation accuracies and losses were obtained so as to show how well trained they were. This is represented in Table 2 where the three models' training and validation average accuracy and loss were presented.

From Table 2, it would be observed that VGG-16 had the best average training and validation accuracies of 0.8946 and 0.9556 respectively, while ResNet gave the lowest as regards the aforementioned metrics, the drastic increase in ResNet validation loss and very low validation accuracy shows it is not well performing at all. ResNet training loss is quite high (1.9929), and the validation loss increased significantly (from 14.6348 to 493.4237) during training. These results indicate that the model did not perform well and struggled to learn the underlying patterns in the data. The extremely low validation accuracy further suggests that the model failed to generalize to unseen data. The same thing is applicable to the average training and validation loss. From this, it obviously implies that VGG-16 was well trained and gave confirming performance accuracy.

The result presented in Table 2 could be buttress with Figure 6 that shows the training accuracies and losses obtained while training the VGG-16 model. This shows the movement in the accuracies and losses over 40 epochs that the model was trained with.

Table 2: Comparison of CNN models' training results

Metrics	VGG-16	ResNet	MobileNet
Training Loss	0.3825	1.9929	0.8354
Training Accuracy	0.8946	0.6298	0.7712
Validation Loss	0.1582	493.4237	0.2109
Validation Accuracy	0.9556	0.0343	0.9546

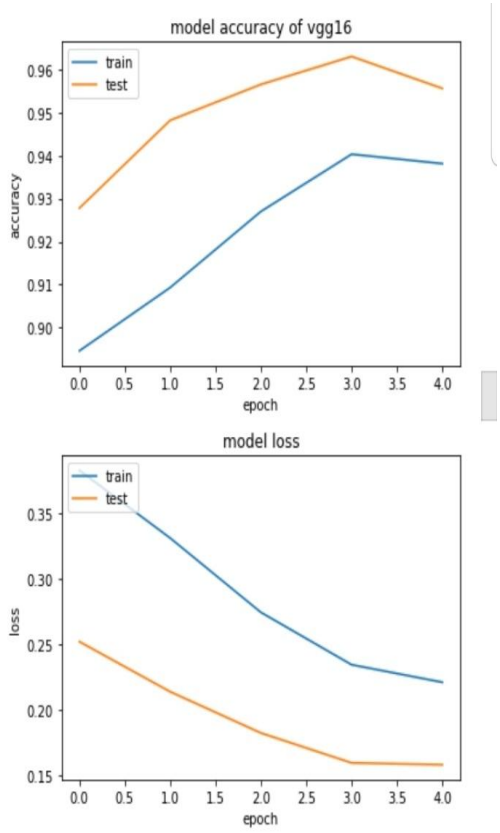

Figure 6: VGG-16 Training accuracies and losses

These models were then tested on the testing set for evaluation. The Experimental result of the models on the testing set is presented in Table 3 for clarification.

As shown in Table 3, VGG-16 and MobileNet models performed well on both the test and evaluation images with average accuracies of 95.6, 97.7, 95.5 and 96.6 respectively, achieving high accuracy and demonstrating good generalization. On the other hand, the ResNet model showed poor performance, with very low accuracy on both datasets. Therefore, the VGG16 and MobileNet models are more reliable for making accurate predictions on unseen images compared to the ResNet model.

From the result shown in Table 2 and 3, it obviously implies that VGG-16 was the best performing model among the three (3) CNN models experimented.

B. Sign language translator mobile app deployment

Figure 7 delves into the introduction of the key details of the application where users see illustrations in carousel format showing that the application helps translate sign language. It serves as a central hub for users to engage with the application's core features. The home screen features a carousel of illustrative content, strategically positioned to capture user's attention upon launch. The carousel design introduces a dynamic element, engaging users with visually appealing graphics that align with the application's theme and purpose. There are other significant elements on the home screen that contribute to enriched user experience: the light-dark mode switch button and a dedicated section showcasing the application's functionalities. The inclusion of a light dark-mode switch button on the home screen provides users with flexibility to tailor the application's appearance to their comfort. All the home screen elements collectively enhance the user experience and provide valuable insights into the application's capabilities.

Figure 8 shows the functionality of the camera screen that enable users to translate sign language gestures representing numbers (0-9) and (a-z). This feature showcases real-time recognition and translation through camera streaming. The camera screen harnesses advanced computer vision and machine learning technologies to process real-time video input from the device's camera. It employs convolutional neural network techniques to detect and interpret users' sign language

Table 3: Comparison of CNN models' testing results

Metrics	VGG-16	ResNet	MobileNet
Accuracy of test images.	95.556%	3.426%	95.463%
Accuracy for Evaluation Images.	97.685%	2.13%	96.574%

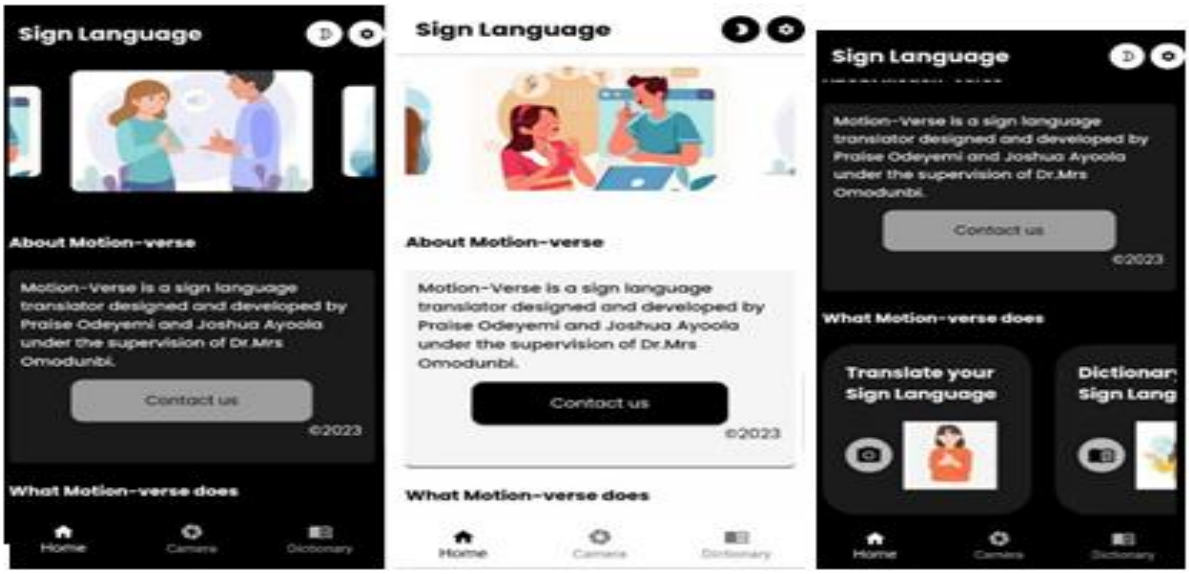

Figure 7: The Mobile app Home Screen

Figure 8: The mobile-app (Motion-verse) Camera Screen

gestures, particularly those corresponding to numbers and alphabetic characters. When a hand gesture matches a recognized sign for a number or alphabetic character, the screen overlays a board showing possible recognitions and their accuracy percentage as recognized using the developed model. The translated content is displayed beneath the camera feed allowing users to see the recognized sign and its corresponding translation simultaneously as shown in Figure 9. The screen is designed to be intuitive and unobtrusive, allowing users to focus on their hand gestures and translated content.

Figure 9 shows the implementation of the sign language dictionary feature in the application. This feature in the application provides a comprehensive collection of sign languages corresponding to A-Z and 0-9 complete with images and details. The data is stored in a database, and the dictionary offers a search functionality for users to quickly find specific signs. The architecture, database integration, user experience, search functionality is discussed. The sign language dictionary leverages the Firebase database to efficiently store and manage the extensive collection of sign languages. The database capabilities ensure seamless updates and synchronization across devices. The search functionality ensures that users can search for specific sign languages by entering characters, numbers, and keywords into the search bar. As users type their search queries, the dictionary dynamically filters entries to display relevant matches in real time. Users can browse sign languages categorized by alphabetic characters. Clicking on a specific sign language entry navigates users to a dedicated screen displaying comprehensive details including images depicting the sign language gesture

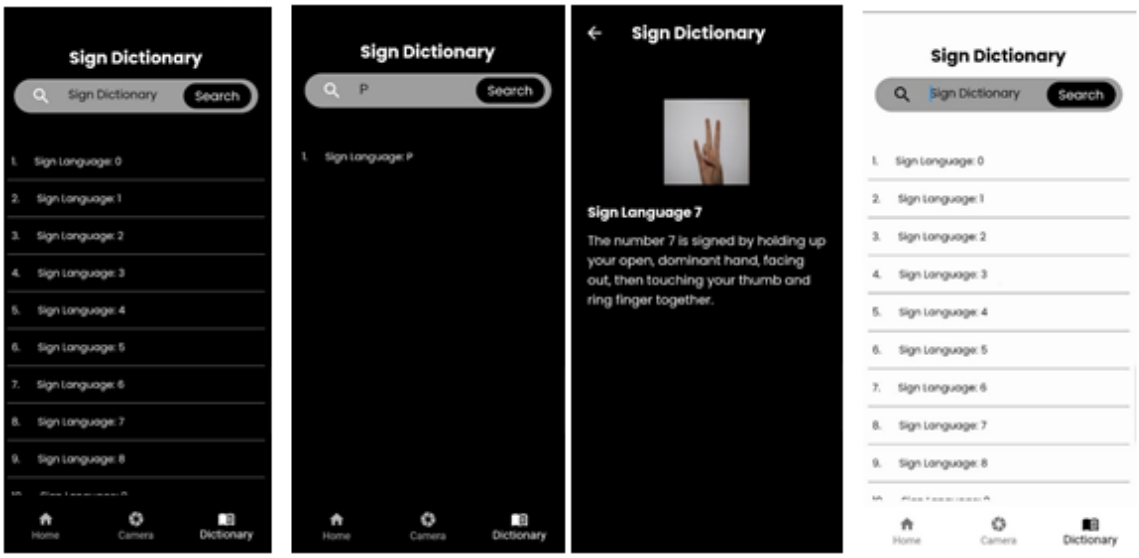

Figure 9: Motion-verse dictionary screen

IV. CONCLUSION

This research has effectively addressed the communication challenges experienced by the deaf and hard-of-hearing community, stemming from the lack of sign language proficiency among non-sign language users. The central objective of creating a mobile-based sign language translator using convolutional neural networks (CNNs) technique has been successfully achieved. The significance of this project extends to its potential to bridge the communication gap between sign language users and the broader community. By harnessing the capabilities of Convolutional Neural Networks and Natural Language Processing, we have managed to develop a real-time and precise sign language translation system that operates seamlessly across both Android and iOS platforms. Future work(s) should focus on inclusion of diverse and extensive collection of English words and

expanding the system's scope to be able to translate multiple sign languages would significantly enhance its utility, while the present version concentrates on Android devices, it is advisable to extend the availability of the sign language translator to iOS users.

REFERENCES

[1] Núñez-Marcos, A.; Perez-de Viñaspre, O.; Labaka, G. "A survey on Sign Language machine translation". *Expert System Application*. Volume 213, Number 118993, 2013

[2] Zhou, H.; Zhou, W.; Zhou, Y.; Li, H. "Spatial-temporal multi-cue network for continuous sign language recognition*". In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12, 2020*, Volume 34, pp. 13009–13016.

[3] Cui, R.; Liu, H.; Zhang, C. "Recurrent convolutional neural networks for continuous sign language recognition by staged optimization". *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA*, 21–26 July 2017; pp. 7361–7369.

[4] Cihan Camgoz, N.; Hadfield, S.; Koller, O.; Bowden, R. "Subunets: End-to-end hand shape and continuous sign language recognition". *In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy*, 22–29 October 2017; pp. 3056–3065.

[5] Koller, O.; Zargaran, S.; Ney, H.; Bowden, R. "Deep sign: Enabling robust statistical continuous sign language recognition via hybrid CNN-HMMs". *International Journal of Computer Vision*. Volume 126, 2018. pp. 1311–1325

[6] Liang, Z.; Li, H.; Chai, J. "Sign Language Translation: A Survey of Approaches and Techniques". *Electronics*. Volume 12, Number 2678. 2023. https:// doi.org/10.3390/electronics12122678

[7] Amusa, K. A., Olanipeku, A. J., Erinosho, T. C., and Abiodun Akeem Salaam, S. S. Development of a PC-based sign language translator. *International Journal of Informatics and Communication Technology (IJ-ICT)*, 2022 pp.23-31.

[8] Yerpude, P., Jagat, P., Sahu, R and Dubey, P. "Non-Verbal (Sign Language) To Verbal Language Translator Using Convolutional Neural Network". *International Journal for Research in Applied Science & Engineering Technology (IJRASET),* Volume 10, Number I, 2022. Pp. 269-273.

[9] Pawar, S., Bamgude, A., Kamth, S., Patil, A. and Barapte, R. "Gesture Language Translator Using Raspberry Pi". *International Journal for Research in Applied Science & Engineering Technology (IJRASET),* Volume *10*, 2022. Pp. 566-570.

[10] Mendhe, M., Bawankule, R., Udapurkar, A., Gogiya, R., Gaikwad, R. and Saggu, J. K. "SignReco: Sign Language Translator". *International Research Journal of Engineering and Technology (IRJET),* Volume: 09, Number 3, 2022. Pp. 328-332.

[11] Valarmathi, V., Sowmiya, S. and Viswanathan, M. EquiSign Dynamic Sign Language Translator. *International Journal of Engineering Research in Computer Science and Engineering. (IJERCSE),* Volume 9, Number 7, 2022. Pp. 13-17.

[12] Khan, W. "A Robust Business Specific Real-Time Sign Language Translator".

International Journal for Research in Applied Science & Engineering Technology (IJRASET), 2022. Pp. 308-311.

[13] Rani, A and Manjanaik, N. "Sign Language to Text-Speech Translator Using Machine Learning". *International Journal of Emerging Trends in Engineering Research,* Volume 9, Number 7,2021 pp. 912 – 916.

[14] Bhavadharshini, M., Josephine, R. J., Kamali, M., Sankar, S. and Bhavadharshin, M. "Sign Language Translator Using YOLO Algorithm". *Advances in Parallel Computing Technologies and Applications*, 2021. Pp. 159-166.

[15] Rahman, M. M., Islam, M. S., Rahman, M. H., Sassi, R., Rivolta, M. W. and Aktaruzzaman, M. "A New Benchmark on American Sign Language Recognition using Convolutional Neural Network". *In the proceedings of the 2019 International Conference on Sustainable Technologies for Industry 4.0 (STI*) December, Dhaka. 2021. Pp. 24-25