EPiC Series in Computing

Volume 88, 2022, Pages 28–37

Proceedings of 31st International Conference on Software Engineering and Data Engineering

IASLS a Mobile Application for Facilitating Communications in American Sign Language

Ian Applebaum[1] Tarek Elseify[1] Liam Miller[1] Likhon Gomes[1] Viet Pham[1] Shakeel Alibhai[1] Aidan Loza[1] Tamer Aldwairi[1,*]

[1] Temple University, Philadelphia, PA 19122, USA

ian.tyler@temple.edu, telseify@gmail.com, liammiller@temple.edu, tug60499@temple.edu, viethpham@icloud.com, shakeel.alibhai419@gmail.com, aidanloza@gmail.com, aldwairi@temple.edu

Abstract

Intelligent American Sign Language System (IASLS) is a mobile application that allows users to operate their phone or tablet's camera to capture American Sign Language (ASL) and convert it into text which could later be saved as a note or copied into a text message or document. Text captured from translated finger spelling or full words in ASL can be used to send a message cross-platform to other IASLS users, to take notes in the app, and to help support an in-person conversation between users with ranging abilities. Additionally, users who do not know ASL can use the app's built-in dictation feature to communicate with people with deafness. IASLS is a mobile application that is designed to facilitate and speed up communication between parties using ASL.

1 Introduction

Communication is an important aspect of life for most people we use verbal words as our main way of communication, but not all people can do so due to issues related to muteness, deafness, or both. The causes of deafness can be genetics, complications at birth, infectious diseases, chronic ear infections, use of drugs and toxicants, excessive noise exposure, and age, while muteness can be caused by endotracheal intubation, tracheostomy, or damage to the vocal cords from disease or traumas. Muteness in a way can also be an effect of deafness. Currently, about 466 million people worldwide require rehabilitation to address their hearing loss, 34 million among which are children, and by 2050, over 700 - 900 million people are estimated to have hearing loss. These numbers demonstrate the deepness and

* Corresponding Author

IASLS is also known as iASL

the magnitude of the problem, but this is only looking into half the problem. There is nearly the same number of people suffering from the ability to lose speech. Even with the advancement of science and tools to aid those people, there are still a lot of people whose only method to communicate is through sign language.

Using sign language enables the deaf and the mute to communicate effectively and provides a global way for everyone to communicate with them accurately and effectively. The problem is that not everyone can use sign language which is why in many cases there is a translator for them especially when they speak publicly or need to communicate with someone who is not familiar with the sign language. That is why it is important to design a tool that can effectively catch the person communicating and translate it into words in real-time so that people can communicate effectively and easily with each other. Through the Utilization of machine learning techniques we can build better tools for such purpose. Previous studies utilizing machine learning techniques [1-22] have shown great promise in this area of research. In this paper, we developed a mobile application called IASLS to help support in-person conversations between users with ranging abilities using American sign language.

IASLS takes the captured visual data from the phone's camera and sends it to a machine learning module (e.g., TensorFlow), which continuously learns, analyzes, and makes decisions based on a given dataset. Results are sent back to the user's device and saved as a text document. Users can use this application for multiple purposes. IASLS can act as a "keyboard" for people who have difficulties typing on their phone, or as an American Sign Language learning tool. Users are also able to communicate with others directly through the app. By using our ASL translating functionality, users can sign directly into the phone's camera and the app can distinguish what the user is trying to say. Once the user has text in place, the user can send their signed message to another user in a private messaging system.

IASLS can run on both the Android and iOS platforms. The iOS version is developed in Swift, while Java is used for the Android version. Both platforms used a unified machine learning component (TensorFlow) for pattern recognition of hand gestures. Specifically, a CNN-LSTM model was used for video classification. The deep learning model can distinguish between many different pre-defined words based on the dataset used. Both Android and iOS workflows work in the same way. A stream of images from the camera feed is pre-processed and sent to the trained model on the device for classification. The predictions are post-processed, and the most confident prediction is displayed in the text field. The aggregate of text is saved on the user's local device.

In order to recognize the gestures efficiently, the machine learning model needs a large dataset of videos/pictures of American Sign Language for training. The training process demands a lot of processing power. Therefore, this phase was done on a machine with an NVIDIA GeForce RTX 2080 GPU. The actual machine learning algorithm consists of a CNN-LSTM architecture. Convolutional Neural Networks (CNNs) are popular in the field of image classification and computer vision. They are used to extract features from an image and create a feature map. Long Short-Term Memory (LSTM) on the other hand is a type of recurrent neural network specialized in learning long-term dependencies. Alternatively, we experimented with other deep learning architectures to achieve peak performance. We experimented with different architectures to find the best-performing model. One such alternative involved using a 3-dimensional CNN which allowed for extracting features in both the spatial and temporal domains. Many iterative experiments were performed to optimize the model. We found the CNN-LSTM model performed best on our validation set.

2 System Block Diagram

Figure 1 illustrates IASLS's system block diagram; images are captured from the user's smartphone and captured as input to our machine learning model. The machine learning model is then diagrammed on the right showing images taken by the smartphone sent as 32x64x200x200 (batch of 32 64-frame video clips) through a 3D-CNN LSTM system with an MLP classifier. After classification, the prediction of the word or letter is sent back to the user's smartphone as text output.

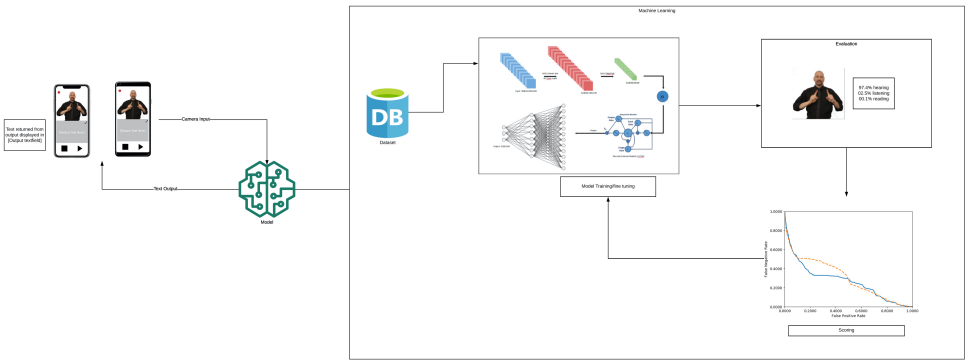

Figure 1: System Block Diagram

3 State Diagrams

Figure 2 illustrates IASLS's Translation State Diagram. It describes the various steps around the app and how to go back and forth in the process. The whole diagram has been color coded to represent different types of components of the app. The color codes are. Violet: View-Controller, Light Purple: Firebase Service, Orange: OS System Call API, Gray: IASLS Translation Algorithm.

At first, the user is prompted to sign into the app, if the user does not have an account, they can proceed to sign up. The sign-in process is only done only if the user is not already signed into the app. Once signed in, every time the user signs in afterward, they can skip the authentication step. The user also has the option to skip signing in and to use the app as a guest. Signing in is required for messaging capabilities.

Once the user is authenticated, they will be shown the main view, which comprises a camera view. The user can use the main feature of the app, the ASL recognition and speech-to-text feature in this view. The camera captures the video using the front camera, it can also be switched to point and shoot using the rear camera. Once the video is recorded, it's sent to the backend server to run by a model. Once the model analyzes the video, it returns the text of ASL in the video, then sent back to the user which is then shown on the display as text and played as a sound using AVSpeechSynthesizer. In order to hold a natural conversation, AVSoundRecorder records what the other person says, and then transcribes it to text, which is then made visible to the ASL speaker on the screen.

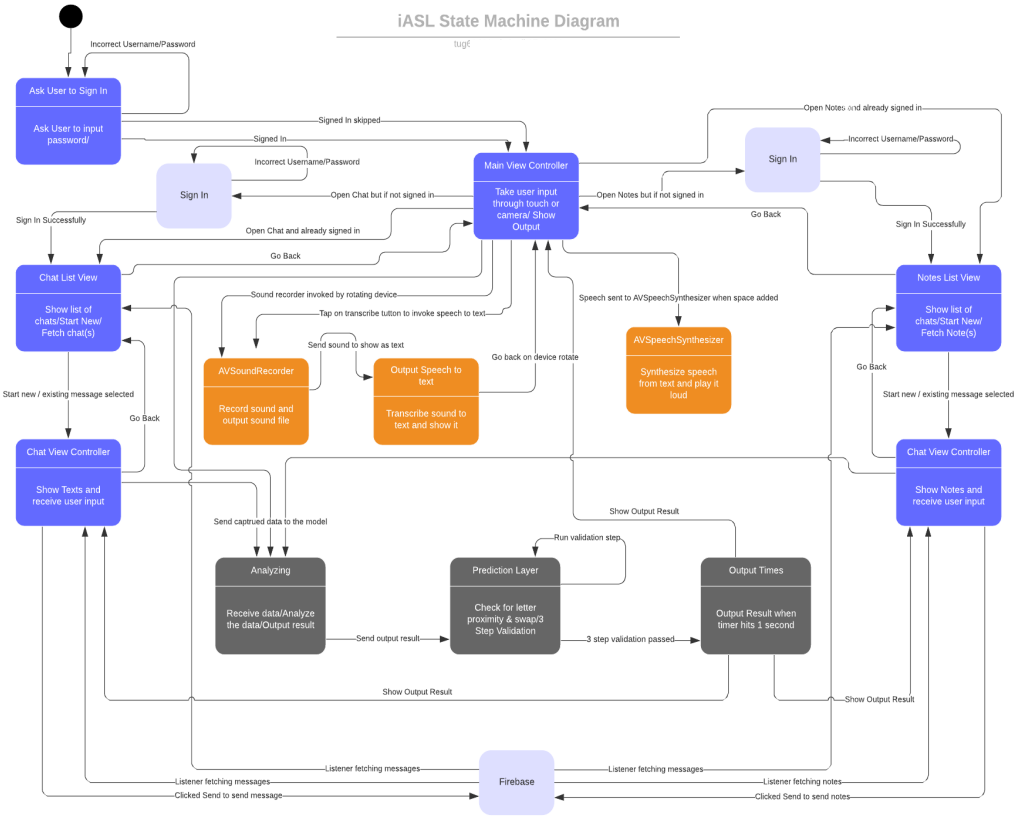

Figure 2: State Machine Diagram

4 Activities

This project requires access to both iOS and Android development environments that are XCode on macOS and Android Studio. Mobile devices are needed for testing, including an iPhone and an Android phone. In order to create and train our machine learning model, we used a remote GPU server with high computational power since our training dataset is very large in size and quantity. Internet access is required in order to connect the application with the back-end server. GitHub was used for project management.

We used TensorFlow software, for both iOS and Android, to help with developing our app. TensorFlow was used as a base to kickstart our implementation of the machine learning aspect of this project since we need to train the app to understand a wide range of hand motions. In order to train the model, we need a large dataset of sign language examples that we can parse and evaluate. The dataset has video clips of someone signing a word or phrase along with a text version of the word or phrase.

Detailed Design:

(1) Convert the MS-ASL dataset into usable data

a. Download YouTube videos

b. Process videos using MS-ASL requirements (bounding box, time stamps)

c. Maintain 1-to-1 correspondence between videos and labels

(2) Develop code responsible for data wrangling and training

a. Generator class used for loading data in batches (can't fit into mem)

b. Normalize videos to fit the model (i.e.: reshaping, grayscale, shortening, extending based on the length of the video

c. Develop different types of ML algorithms (3d-CNN, Inception 3d, CNN-LSTM)

(3) Develop code for evaluating a model

a. Decoding script used for loading a test dataset

b. Scoring the output of the saved model against ground truths of the test set

(4) Perform tuning to find optimal model parameters for best performance

(5) Launch Camera Activity and hold X + K number of frames from the stream, normalize the FPS among devices to ensure the same frame rate

a. X is the number of frames the model takes in, K is the number of buffer frames

b. The idea is that if X is 30 frames (i.e., one second), a signed word may not be in the same exact time as that one-second interval. So, a buffer 11of frames are used, and a sliding window technique was used to predict every 30 frames of the image stream. The predictions are post-processed so that once the threshold confidence has been reached (I.e.: 80% confident), that is the word that is the output. This means the model will run a bit slower than in real-time.

(6) Use Firebase Realtime Database to send messages between users.

(7) For Android, the 'Play' button activates TTS by invoking its built-in Text-To-Speech engine. This will concatenate all the output of the model into a string and pass it to the engine for playback. For the iOS version, the same process is used and Apple's Voice Over API will convert the string to speech.

(8) The phone's microphone records the non-mute users' voice and the app's speech-to-text function can convert the recording to text and send it over the web to a hearing-impaired user.

5 Development Environment

The required development environment is two-fold. For the iOS version of our project, we used XCode to develop our project. In XCode, we used Swift to write our code. For the Android version of our app, we used Android Studio as our base, and we developed the layout and functioning of the app with Java. In addition to Java, we use XML to design the look and feel of the app. The base of our app is built on software already produced by Google's TensorFlow. TensorFlow is an open-source collection of software that helps with building machine learning platforms. Since TensorFlow already has some of the camera and text displaying functionality ready to go, we use this to kickstart the design of the app and help use it to create our sign language recognizing machine learning model.

In order to test our applications, we used the simulators that come with XCode and Android. Both have decent simulators that are very helpful in creating and testing the design of the app, however, the simulators have their limits. Since our app is heavily based on the use of the phone's camera, and the simulators do not allow developers to use the computer's camera, we used our phones to test the camera and motion recognition parts of the functionality. Our team has an equal distribution of users with iPhones for iOS testing and Android phones for Android testing, so this should not be a disadvantage. We used a software called Telegram for the messaging part of our app. Telegram has an API that we for sending and receiving messages from other users. We used this to integrate the ASL translations.

In addition to the environments, we used a GPU server to download an ASL dataset that has been collected by Microsoft. The dataset contains a significant number of words and a video of their translations in American Sign Language. After that, we parsed the collection of words and used the

videos to create our machine learning model. After downloading this dataset, we created a Python script to parse through this data and create training, testing, and validation sets, as well as normalizing all the videos to a 200x200 frame with a total length of 64 frames. Downscaling all the videos to 200x200 certainly caused a loss of resolution, however, this tradeoff was worth it as higher dimensions would drastically slow the training process.

6 Algorithms

As we have a machine learning backend, we implemented a neural network and fed it training and testing data. We created a 3-dimensional convolutional neural network paired with a Long Short-Term Memory (LSTM) recurrent neural network. The 3-dimensional convolutional layer generates spatiotemporal features for each video input. These features were passed along to the LSTM layer where long-term dependencies were learned. Finally, the outputs of the LSTM were flattened and passed along to a fully connected multi-layer perceptron. This represents the classification portion of the model and has an output layer of 1000 nodes, one for each word in the dataset.

We downloaded pre-categorized videos of various examples of signing. These videos were trained on the neural network over 10 epochs. We ran the training data through enough training epochs for the neural network to learn the various features of the signing data; however, we ran into system limitations as running the data through many epochs required the model to run on the GPU for an extensive amount of time. Fortunately, this wasn't an issue as we found that after about 7 epochs the model began overtraining and performed worse on validation sets.

We implemented certain techniques into our neural network to reduce the overfitting problem, which may occur if the neural network simply "memorizes" the training data instead of learning its features. If the former occurs, then the neural network would likely perform very well on the training data but would likely perform much worse on the testing data. In order to mitigate the problem of overfitting, we utilized a dropout technique. The technique is a way of reducing overfitting by randomly ignoring a given percentage (50% in our case) of neurons from the training step in each epoch. This ensures that the model is less reliant on specific neurons and can assign appropriate weights to all neurons. Another way of eliminating potential overfitting could be to simply ensure that the number of epochs is not too high. We trained the model for 10 epochs and saved the weights after each epoch. After training was complete, we loaded the model with each of the weights and ran them on our validation set. From there, we simply chose the weights that had the highest accuracy for the validation set.

7 Database Design

Figure 3 illustrates the design of our database that we used to store the information for our messaging and notetaking features. Firebase uses a node tree to store its data, so this is the best representation of that, however, there are no foreign keys or primary keys in a way. Those are defined in the code when go and read and write to the tree of nodes. In theory, in each of these tables, there is a node in its tree and the values inside each one is a sub-node of the root node. The "Users" node/table holds a unique ID of each user. It also holds the name and the email of the user. These values are obtained by the authentication feature that Firebase provides. In the "Messages" table, there is a unique ID for each message sent. Each message also has a receiver ID and a sender ID that correlate to the "Users" table.

There is always a sender and a receiver in each message sent. Each message node also has a text field that holds the text of the message as well as a timestamp of when the message was sent.

There is also a “User-Messages” node/table that helps with identifying which user owns which message. This is mainly here to quickly and easily track down every message in our messages table that belongs to a particular user. The strategy for this is called fanning out. Instead of observing every message in the table when we go to gather someone’s messages, we can keep track of which messages they own via the user-messages table, which holds the IDs of every message that every user holds, and then use those IDs to track down the messages in the messages table. This will save time and money so that we do not have to read from the database as much.

On the notetaking side, the concept is pretty much the same. Each notes table holds a unique “Notes” ID, the owner ID which correlates to someone in the “Users” table, the text field which holds the text of the note, the title field which holds the title of the note, and the timestamp which holds the time it was created or updated. The “Notes” table also has a “User-Notes” table which does the same kind of user tracking as the “User-Messages” table does. When a note gets made, it finds or creates a node with the ID of the user and saves every note ID in that node for quick access to who owns which note.

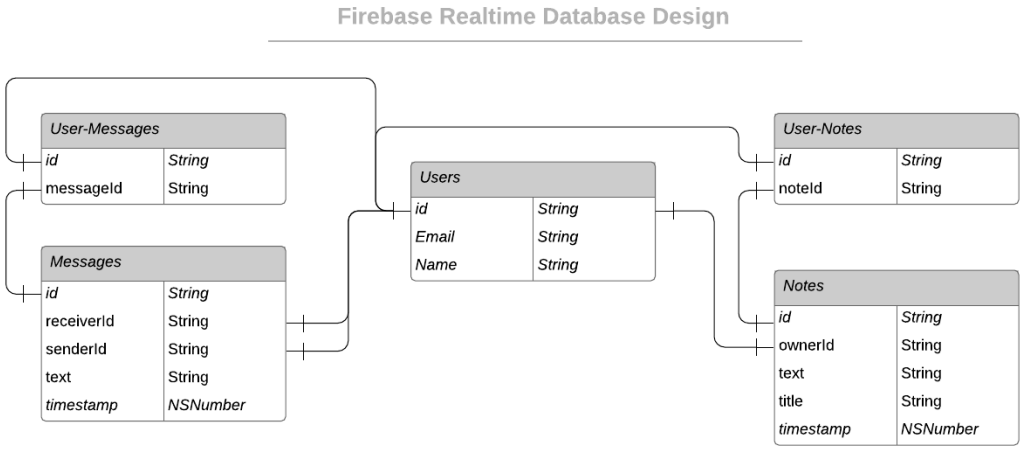

Figure 3: Database Diagram

8 Testing

For this project, the model training phase takes a large amount of time. This prompted us to use at the beginning only a portion of the data to create the test model to generalize the data and to make sure our machine learning implementation is accurate. We then moved on to the actual training phase of the model with all the datasets.

Next, to test the mobile application we used GitHub’s continuous integration platform GitHub-Actions, so we can test our application for every merge into the main branch. These mainly comprised of end-to-end tests of the user interface where user interactions such as button presses were simulated. Unit testing of the app consisted of testing the connection between our software and the ASL recognition model. We sent simple consistent images to our ASL letter recognition API and would assert the

expected letter output. There were aspects of the application that were not able to be captured using automated testing. Specifically, the smartphone's video capture could not be easily or reliably tested using end-to-end testing, so we opted for manual acceptance testing for these features. Our team would manually pull in any new code to our local machine and install the testing version of the application on our physical smartphones. This testing helped ensure that our project's main branch was always relevant and never broken. Therefore, if the main branch ever gets corrupted by a faulty pull request, GitHub's integration testing will alert us about it.

There were issues with using GitHub Actions for continuous integration. The speed of GitHub macOS virtual environments was limited and often very slow. This made testing the iOS Application much slower than testing the Android equivalent as the Java environment can run in GitHub Linux virtual machines at a faster rate. We are also limited to only 5 concurrent macOS testing environments for our iOS application whereas our Android application can have up to 20 concurrent testing environments.

9 Conclusion

In this paper, we introduced a mobile application called IASLS that allows users to operate their phone or tablet's camera to capture American Sign Language (ASL). The application has a lot of capabilities, it allows for fast finger spelling detection and is very responsive to ASL word detection. The user is provided the ability to help in training the video classification model. In addition to a multi-platform chat feature between IASLS users and a note-taking feature. The application has still ways to go before coming to a full-fledged software tool but in the meantime, the application provides the users with a very convenient way to communicate with each other without the need for one of the parties to have a translator or the other party to know about ASL. Future work might include building the tool as part of a pipeline [23-28, 30-32] of programs to provide more functionality to the users or designing another program using the same factors [29] and inserting it within the tool to perform additional functionality.

10 Resources

IASLS Mobile Applications version for both Android and iOS can be found at the links below
https://github.com/Capstone-Projects-2020-Spring/iASL-iOS
https://github.com/Capstone-Projects-2020-Spring/iASL-Android

The Algorithmic detail for the IASLS backend is available at https://github.com/Capstone-Projects-2020-Spring/iASL-Backend

References

[1] Kumar P, Gauba H, Roy PP, Dogra DP. Coupled HMM-based multi-sensor data fusion for sign language recognition. Pattern Recognition Letters. 86: p. 1-8, 2017.

[2] Jin CM, Omar Z, Jaward MH. A mobile application of American sign language translation via image processing algorithms. Proceedings of IEEE Region 10 Symposium, TENSYMP. p. 104-109, 2016.

[3] Cayamcela MEM, Lim W. Fine-tuning a pre-trained Convolutional Neural Network Model to translate American Sign Language in Real- time. 2019 International Conference on Computing, Networking and Communications, ICNC. p. 100-104, 2019.

[4] Xu P. A Real-time Hand Gesture Recognition and Human-Computer Interaction System. Apr 24, 2017.

[5] Pigou L, Dieleman S, Kindermans PJ, Schrauwen B. Sign Language Recognition Using Convolutional Neural Networks. In Pigou L, Dieleman S, Kindermans PJ, Schrauwen B. p. 572-578, 2015.

[6] Hore S, Chatterjee S, Santhi V, Dey N, Ashour AS, Balas VE, et al. Optimized Neural Networks. p. 139-151, 2017.

[7] Abdelnasser H, Harras KA, Youssef M. WiGest demo: A ubiquitous WiFi-based gesture recognition system. Proceedings - IEEE INFOCOM. p. 17-18, 2015

[8] Thongtawee A, Pinsanoh O, Kitjaidure Y. A Novel Feature Extraction for American Sign Language Recognition Using Webcam. BMEiCON 2018 - 11th Biomedical Engineering International Conference. p. 1-5, 2019.

[9] Aly W, Aly S, Almotairi S. User-independent american sign language alphabet recognition based on depth image and PCANet features. IEEE Access. p. 123138-123150, 2019.

[10] Koller O, Zargaran S, Ney H, Bowden R. Deep sign: Hybrid CNN-HMM for continuous sign language recognition. British Machine Vision Conference, BMVC. p. 136.1-136.12, 2016.

[11] Brandon Garcia , Sigberto Alarcon Viesca. Real-time American Sign Language Recognition with Convolutional Neural Networks. [Online]. Available from: http://cs231n.stanford.edu/reports/2016/pdfs/214_Report.pdf.

[12] Taskiran M, Killioglu M, Kahraman N. A Real-Time System for Recognition of American Sign Language by using Deep Learning. 2018 41st International Conference on Telecommunications and Signal Processing, TSP. p. 1-5, 2018.

[13] Guo D, Zhou W, Wang M, Li H. Sign language recognition based on adaptive HMMS with data augmentation. Proceedings - International Conference on Image Processing, ICIP. p. 2876-2880, 2016.

[14] Rao GA, Kishore PVV. Sign language recognition system simulated for video captured with smart phone front camera. International Journal of Electrical and Computer Engineering. 6(5): p. 2176-2187, 2016.

[15] Yeo HS, Lee BG, Lim H. Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware. Multimedia Tools and Applications. 2015; 74(8): p. 2687-2715.

[16] Cui R, Liu H, Zhang C. Recurrent convolutional neural networks for continuous sign language recognition by staged optimization. Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR. p. 1610-1618, 2017.

[17] Huang J, Zhou W, Zhang Q, Li H, Li W. Video-based sign language recognition without temporal segmentation. 32nd AAAI Conference on Artificial Intelligence, AAAI. p. 2257-2264, 2018.

[18] Joshi A, Sierra H, Arzuaga E. American sign language translation using edge detection and cross correlation. Proceedings of IEEE Colombian Conference on Communications and Computing, COLCOM 2017.

[19] Shahriar S, Siddiquee A, Islam T, Ghosh A, Chakraborty R, Khan AI, et al. Real-Time American Sign Language Recognition Using Skin Segmentation and Image Category Classification with Convolutional Neural Network and Deep Learning. IEEE Region 10 Annual International Conference, Proceedings/TENCON. p. 1168-1171, 2019.

[20] Ahmed W, Chanda K, Mitra S. Vision based Hand Gesture Recognition using Dynamic Time Warping for Indian Sign Language. In International Conference on Information Science (ICIS) IEEE. p. 120-125, 2016.

[21] Jalal MA, Chen R, Moore RK, Mihaylova L. American Sign Language Posture Understanding with Deep Neural Networks. 2018 21st International Conference on Information Fusion, FUSION. p. 573-579, 2018.

[22] Flores CJL, Cutipa AEG, Enciso RL. Application of convolutional neural networks for static hand gestures recognition under different invariant features. Proceedings of the 2017 IEEE 24th International Congress on Electronics, Electrical Engineering and Computing, INTERCON p. 5-8, 2017.

[23] Aldwairi T, Chevalier DJ, Perkins AD. Exploring the Effect of Climate Factors on SNPs within FHA Domain Genes in Eurasian Arabidopsis Ecotypes. Agriculture. 11(2):166, 2021. https://doi.org/10.3390/agriculture11020166

[24] Aldwairi, T., Elam, B., Hoffmann, F., & Perkins, A. D. RepCalc: a Tool For Calculating Transposable Element Density within the Genome. Proceedings of the 34th International Conference on Computers and Their Applications (BICOB) 2018.

[25] T. Aldwairi, B. Nanduri, M. Ramkumar, D. Gautam, M. Johnson, A. D. Perkins. "Statistical Methods for Ambiguous Sequence Mappings". In Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics (BCB'13), Association for Computing Machinery, New York, NY, USA, 674–675, 2013. DOI:https://doi.org/10.1145/2506583.2506678

[26] T Aldawiri, et al. "A Novel Approach for Mapping Ambiguous Sequences of Transcriptomes." In Proceedings of 14th International Conference on Bioinformatics and Computational Biology. Vol. 83, pp. 76-85. 2022.

[27] Aldwairi T, Hoffmann F, Perkins AD. Prediction Of Novel Pirna Rat Clusters Based On Mouse Pirna Clusters Using Downstream and Upstream Analysis. InProceedings of 11th International Conference. Mar 18 (Vol. 60, pp. 190-199), 2019.

[28] Aldwairi, Tamer Ali, "Computational Methods for Solving Next Generation Sequencing Challenges" (2014). Theses and Dissertations. 1140.
https://scholarsjunction.msstate.edu/td/1140.

[29] Al-Agtash, Salem Y., et al. "Re-Engineering BLUE Financial System Using Round-Trip Engineering and Java Language Conversion Assistant." Software Engineering Research and Practice. (pp. 657-663) 2006.

[30] Aldwairi T, Perera D, Novotny MA. Measuring the Impact of Accurate Feature Selection on the Performance of RBM in Comparison to State of the Art Machine Learning Algorithms. *Electronics*. 9(7):1167, 2020. https://doi.org/10.3390/electronics9071167

[31] Aldwairi, T., Perera, D., & Novotny, M. A. An evaluation of the performance of Restricted Boltzmann Machines as a model for anomaly network intrusion detection. *Computer Networks*, *144*, 111-119, 2018.

[32] Aldwairi T, et al. An Investigation of the Role of Feature Selection on the Classification Performance of Machine Learning Algorithms. In Proceedings of the 33rd International Conference on Computers and Their Applications (CATA). 2018.