American Sign Language Interpreter: A Bridge Between the Two Worlds

Kanika Sood
Department of Computer Science
California State University, Fullerton
Fullerton, United States
kasood@fullerton.edu

Bhargav Navdiya
Department of Computer Science
California State University, Fullerton
Fullerton, United States
bhargavnavdiya@csu.fullerton.edu

Anthony Hernandez
Department of Computer Science
California State University, Fullerton
Fullerton, United States
ahernandez197@csu.fullerton.edu

2023 International Conference on Artificial Intelligence and Smart Communication (AISC) | 979-8-3503-2230-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/AISC56616.2023.10085066

***Abstract*—American Sign Language (ASL) is the third most commonly used language after English and Spanish. In this work, we build an image classification modeling technique for ASL to abridge the gap between native ASL speakers including children and others. This paper focuses on providing a sign language recognition system using machine learning. We use four conventional machine learning techniques: K-nearest neighbor, Naive Bayes, Logistic Regression, and Random Forest to detect the alphabets from the images made available using an existing dataset and a new dataset that we generate for this work. Our technique identifies images based on the grayscale values, to identify the same sign in different environments such as images captured in different illuminated environments or hand signs placed at different places compared to the image in the dataset, or hand signs with diverse backgrounds. We use an existing dataset and a real-world dataset that we create independently by generating images using an HP webcam using a computer vision library. We use supervised machine learning and train the classifiers using the labeled image data to predict the ASL signed alphabet in the new image. Our analysis indicates that K-Nearest Neighbor performs best with both datasets achieving up to 99% accuracy.**

Keywords-Computer Vision, Data Augmentation, American, Sign Language, Machine Learning, Naive Bayes, K-Nearest Neighbor, Logistic Regression, Random Forest

I. INTRODUCTION

The ability for humans to verbally communicate with each other is an essential and basic skill that many lack. According to the World Health Organization, over 5% percent of the world's population suffers from hearing loss that can be categorized as 'disabling' [1]. Creating an American Sign Language (ASL) classifier can help individuals with difficulty communicating with others, using sign language. An ASL classifier can detect the sign language and show the alphabets to others whose native language is not ASL. In the United States alone, approximately 6.6 million people aged 12 years or older have profound hearing loss in at least one ear – that is approximately 2.5% of the Americans [2]. Furthermore, over 20% of those 6.6 million individuals use American Sign Language (ASL) as their primary form of communication [3]. Figure 4 shows the ASL alphabet chart for all the alphabets in ASL. This type of classifier, which can classify sign language using training data, makes it readily available on devices with low computation power. To assist in eventually bridging the communicative gap between ASL signers and individuals with normal auditory capabilities, we present this preliminary work that aims towards building an ASL classifier which can identify sign language alphabet by alphabet.

This paper is structured as follows: Section II lays the background work in this area. The next section presents details about the datasets used and the data processing steps.

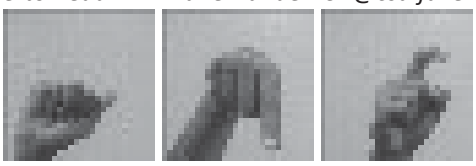

Fig. 1. Sample images for letters A, Q and X (left to right) collected to build a new dataset.

Fig. 2. OpenCV interface indicating the user that the letter to be signed within the square (region of interest) is the letter 'A'.

Section IV presents the challenges faced in this work. In the next section we provide the methodology followed by the results presented in Section VI. In the last section we outline the conclusion and future scope of this research.

II. BACKGROUND

There has been work done in the past to classify images as alphabets for ASL. Some researchers apply graph matching [3] to recognize hand postures, the Gabor Wavelet Transform technique [8], and the F-ratio feature extraction technique [10]. In contrast, some researchers use gloves [10] equipped with electronic devices to detect hand gestures. Some use Infrared Sensors with 3D cameras to get the hand's position, while other researchers use leap motion censors [10] to get accurate hand gestures. Convolution Neural Network (CNN) is a popular approach in the field and researchers use the network to detect the hand position using a camera image, by providing it to a network that has weighted sub-branches which separate the hand from the rest of the background.

Those images are then fed as input to other machine learning algorithm. This is also called a vision-based approach as it only requires a web camera or 3D camera, or color-coded gloves to detect hands.

Google has an API called ”Media-pipe hands” [11] to estimate hand position from a simple web camera. Similar to a face recognition system, this API will assign 21 points to different coordinates of a hand. These 3-dimensional coordinates are then used to calculate the distance between each other and the angle between the points. In order to classify the sign, distance and angle-based feature values and labels are provided to train the model.

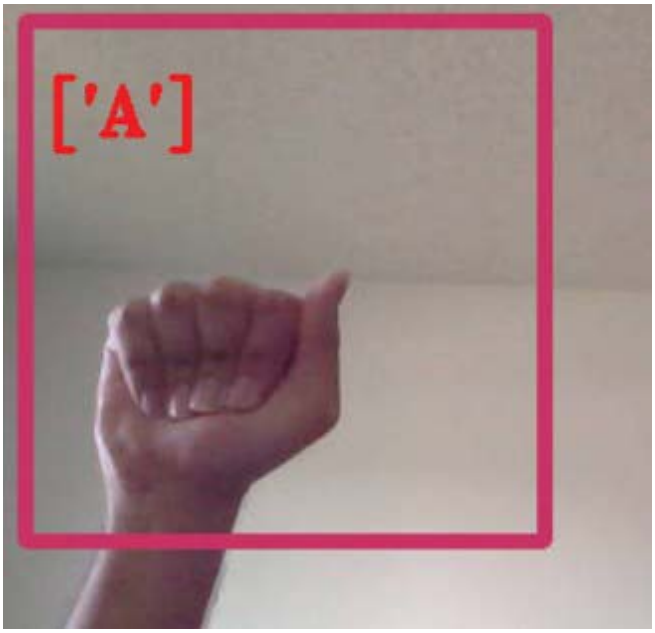

Fig. 3. Classification of the alphabet ’A’ by KNN.

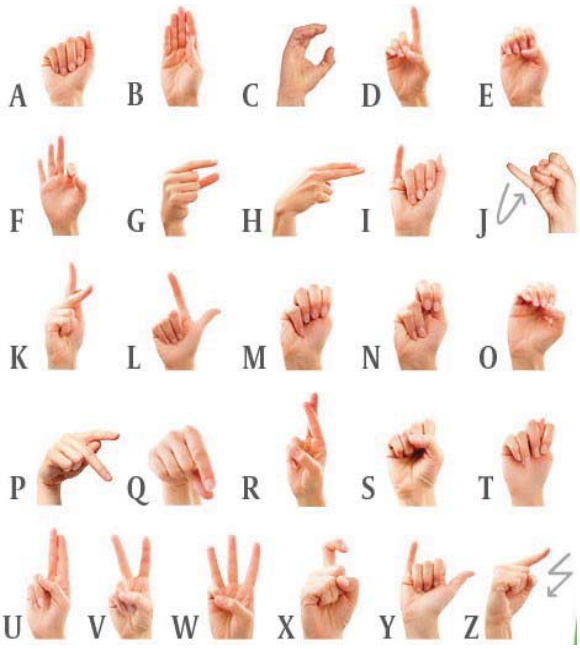

Fig. 4. ASL alphabet chart.

While some researchers use a Leap motion device (a small device with two infrared cameras between three infrared led lights developed by David Holz and Michael Buckwald) [10], the device allows computers to detect fingers, hands, and arm movements in its tiny four cubic feet interactive area. Fingertip and palm position and velocity, including its movement direction and velocity, can be collected to train the model for different sign alphabets. As this model does not depend on any image illuminance or background values, calculating Euclidean distance between points or vectors can easily classify an alphabet.

A depth camera [12] is also a good option to detect hand gestures, including hand detection, orientation normalization, and feature extraction. The distance from the gesture centroid to the left, right column, and the top row is calculated to get the boundary containing the hand. This square is divided into four sections which are further divided into four subsections. Subsections are defined with zero values when empty empty. This process of creating and removing empty subsections are repeated four times, and the final subsections are used as an input to train and test the model.

As described above, the idea of using machine learning to classify the ASL alphabets is one that is not uncommon – the fact that there are many readily available datasets for this task on online Data Science and Machine Learning platforms, such as Kaggle [10], is a testament to this. In this work we use a variety of ML techniques to classify

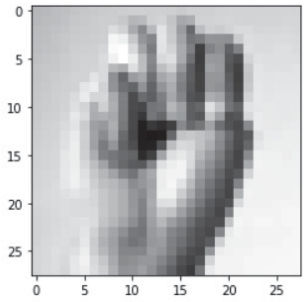

Fig. 5. Sample image of the MNIST dataset.

ASL alphabets well. The main contributions of this work include:

1) *Perform analysis using a well-established existing dataset- MNIST to build a well-performing classifier to classify ASL alphabets.*

2) *Generate a new dataset using a low-profile machine to represent the real-world images containing ASL alphabets. Sample images for alphabets A, Q and X generated and saved in this dataset are shown in Figure 1.*

3) *Build a machine learning classifier that can perform well for the generated dataset.*

4) *Analyze different machine learning techniques and compare their performance for the MNIST dataset and the new dataset generated in this work to see if the optimal ML model stays consistent for similar content, but differently generated datasets.*

Given a pool of readily available datasets, we choose an existing dataset: MNIST for ASL image collection. However most of the existing datasets consist of images that are different than images that are captured by a live camera. Hence we also generate a new dataset with the OpenCV vision library [9]. As shown in Figure 2, OpenCV guides the user to input the sign in a specific region so that it can identify the alphabet. Figure 3 shows the box region provided by OpenCV after the user has provided the input identified as the alphabet ’A’ by one of the classifiers used in this work: KNN. A

continuous loop is used to read-through and capture the frames from the video feed of an HP laptop webcam. The webcam specifications are as follows: HP

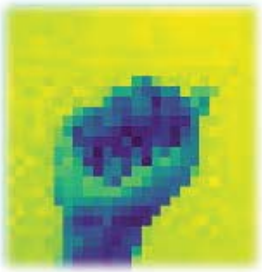

Fig. 6. An image of 'A' after a few transformation such as converting to 28x28 pixels and converting it to grayscale, applied to it.

Truevision HD webcam with 17 FPS, aspect ratio of 1.78, 2.38 MB/s bitrate, 89,026 number of colors with an image mode of RGB Color, and a resolution of 1280∗720. The lightness of the webcam is 24.71%, luminosity of 25.83%, 24.44%, hue at 102◦ and a saturation of 7.94%.

III. Dataset And Data Processing

We have used basic laptop front screen camera to create the dataset, and upon a key-based trigger, a portion of the overall frame – or 'region of interest' – is saved to memory. On the live feed, we place text indicating the current letter that should be signed within the region of interest as shown in Figure 2, of which is based on checking for whether a local directory for that particular letter exists. Then, generally, when the signed letter is within the region of interest and the user is ready to begin taking images. The user can press down on the designated trigger key, and photos are taken until a preset quantity is reached (Script will allow the user to take 200 pictures for each letter, in this case). After the limit is achieved, the screen shows a signal that a new letter is about to appear. This process repeats until there are no missing directories found. Figure 5 shows the alphabet 'A' signed and saved in the MNIST dataset.

The images we have collected using the camera contain high pixel and RGB color values and more information, which requires more computation by the algorithm. So in the next task, we convert those pictures to fewer pixel images and grayscale so that processing that photo by the algorithm makes it less complex and increases the speed. So, before images are saved, they are rescaled to 28x28 pixels using OpenCV's resize function [9] and convert the color format of the images to grayscale format to save space and reduce the processing time later. The activity of creating this first, base dataset results in approximately 4,800 images: 200 images for each English language alphabet, excluding the alphabets 'J' and 'Z' as they require non-static signing that is, signs which involve hand motions. Then, to further expand this collection of images, we use a data augmentation library called 'imgaug' to create new images that will be saved into memory and to create images that would only exist during runtime. In the case of the former, since all of our pictures are signed exclusively with the left hand, we create a mirrored version of each of those photos to simulate having taken them right-handedly: this creates 4,800 additional images that are saved into local storage as right-hand signs are also important because most of the people have their right hand as their dominant hand. To increase the variation between images, we can also use both hands while taking pictures for each alphabet. Because for both hands, the background and lighting environment will vary, which will cause different grayscale values than taking all pictures using one hand and then modifying them using the imgaug [8] library. During run-time, for each of these 9,600 photos, we create additional photos by applying various transformations such as: adding gaussian noise, rotation, cropping, and cutting-out small parts of images altogether under a random range. This is done by using some of the functions available through imgaug. Fig. 6, 7 and 8 show the transformations applied to alphabets A and B respectively.

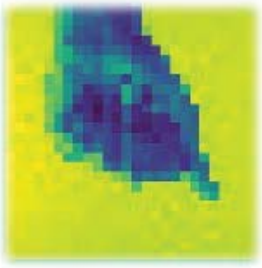

Fig. 7. A second image of 'A' with some transformations applied to it.

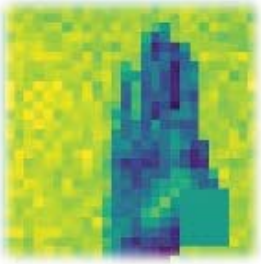

Fig. 8. An image of 'B' after transformations applied to it.

The quantity of images generated per photo in memory is partially dependent on the model used in machine learning. Algorithms such as KNN (K-Nearest Neighbor) work relatively fast, so generating 15+ images for the 9,600 images is feasible. However, other algorithms such as Random Forest are comparatively slower due to the creation of multiple decision trees for each alphabet, hence fewer images are fed into models like this. The structure of the decision tree depends on the value of particular features, but here the features are the grayscale value of pictures, which contains replicated blurry and cropped images causes to produce false results for the given prediction value of the grayscale image.

IV. Challenges

In this section we present the challenges faced in this work. Firstly, if we use an existing database, it results in having images with different illuminance environments and backgrounds with the variation of hand sign gestures. This can impact the performance of the ML models. Secondly, capturing an image using a web camera produces a high number of pixels, hence conversion to grayscale and processing each image by their grayscale values and classifying them using that takes much longer time than expected. Additionally, signing the same alphabet with the location varied can cause different grayscale values, which can also impact the performance of the models. Hence a solution to the aforementioned concerns are resolved by taking pictures from webcams and manually labeling them and creating our own database ensures that the overall

language interpreter performance does not suffer due to inconsistencies even at minute levels.

V. Methodology

In this section, we present the technique we use for building a well-performing ASL Alphabet Classifier using machine learning techniques that serves as a language interpreter. We set up an individual directory for images for each of the twenty-four alphabets separately. We start by looping through these directories to extract all the photos stored in the directories. Every time the user clicks an image using a camera, we make a copy and add it to the database alphabet list, along with the alphabet value represented by that image, into a separate list. We use this database created by individuals to train the ML model and improve its performance.

While collecting images, we use a loop to take up to 200 pictures per alphabet and use imgang function to create slightly altered pictures (add noise, blur, flip, and change contrast) to train the model well. These images are classified into sub-folders according to the alphabet they represent. After creating a dataset for all the 24 alphabets, these images are converted to NumPy [14] arrays based on their labels. As most ML algorithms perform well with one-dimensional data, we reduce the dimensions of the two dimensional images to get one-dimensional NumPy arrays.

One such model we use is the Ensemble Learning Method, Random Forest. We convert the target categories into numerical values by using the factorize method offered by the Pandas library [11], which also returns the sorted categorical labels. Next, we run scikit-learn's [12] traintest-split function on the images array and numerical labels and then use the resulting training data to fit a Random Forest Regressor with 80 trees (number of trees chosen arbitrarily), subsequently using the test images to predict the labels. The predictions we obtain are continuous values, so we use a loop to round them and convert the values back into categorical values with NumPy's vectorize function. In addition to Random Forest, we use three other types of algorithms: Na¨ıve Bayes, Logistic Regression, and KNearest Neighbors. As a base level to test the models, during runtime we set the image factor equal to 10 which means, 10 augmented photos will be generated per image in the dataset. During evaluation the ML models that execute fairly quickly, we further increase the image factor to 20 to enhance the classifier performance.

For analysis, the tools we use are: scikit-learn's confusion matrix, scikit learn's classification report, a data visualization library called Seaborn [13], and a helper function we implement to determine the per-class accuracies of the twentyfour alphabets we examine. This helper function is used to identify which alphabets are generally the most difficult to classify and which ones are relatively easier, on a per-model basis. We determine this by iterating through the rows of the confusion matrix generated with scikit-learn, summing the row values, and then dividing the true positives present in that row by the calculated sum.

VI. Results

In this section, we present the results and finding of this research. We provide the performance analysis for both the datasets that are used throughout this work: (1) A preexisting dataset from the Modified National Institute of Standards and Technology database (MNIST) database for 24 alphabets excluding the non-static alphabets 'J' and 'Z'. (2) A self-created dataset that is generated by signing the 24 alphabets in ASL and taking multiple images using a basic camera offered by a low-profile laptop available for use during this work. In this section, we present the comparative analysis for both the datasets and use multiple machine learning techniques to identify the best performing technique across the two databases using five performance metrics: confusion matrix, precision, recall, F1-score and overall accuracy.

TABLE I. Performance Metrics For The Classification Models For Sign Language Mnist Dataset

Models	KNN	Naive Bayes	Logistic	Random Forest
Precision	0.99	0.45	0.95	0.99
Recall	0.99	0.50	0.95	0.99
F1-Score	0.99	0.45	0.95	0.99
Accuracy	0.99	0.45	0.95	0.99

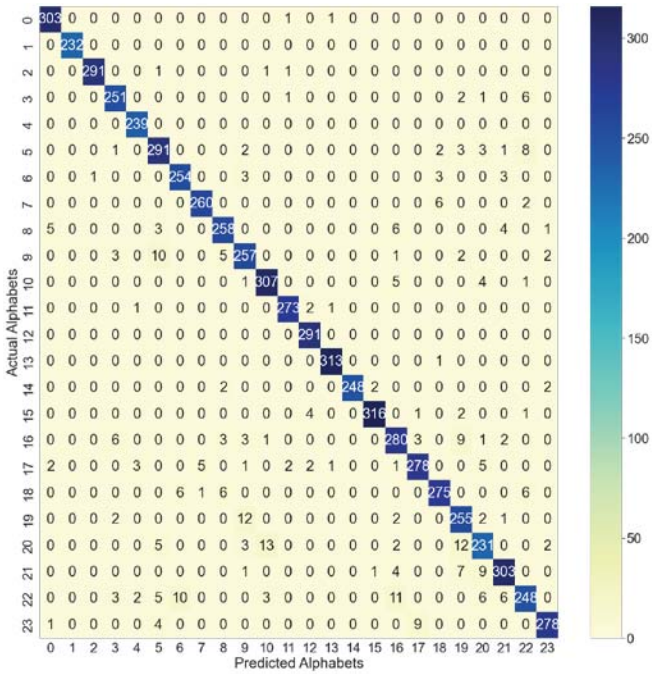

Fig. 9. Confusion Matrix for the test data for all the 24 alphabets A- Z represented as alphabets 0-23, for the MNIST dataset with the best performing classifier: KNN.

Accuracy Score (Range 0-1) of the Ten Most Accurately classified Letters on MNIST dataset							
KNN		Naïve Bayes		Logistic Regression		Random Forest	
H	1.0	B	0.478827	H	0.970149	R	0.997442
G	1.0	Q	0.527704	Q	0.975309	A	1.000000
O	1.0	G	0.571429	P	0.976378	P	1.000000
P	1.0	O	0.597701	M	0.985560	K	1.000000
F	1.0	L	0.611702	C	0.989796	H	1.000000
T	1.0	I	0.675000	A	0.993443	F	1.000000
D	1.0	E	0.684588	O	0.996815	C	1.000000
C	1.0	P	0.723270	N	1.000000	B	1.000000
B	1.0	T	0.724432	E	1.000000	Q	1.000000
K	1.0	C	0.764179	B	1.000000	Y	1.000000

Fig. 10. Most Accurately Classified Alphabets for the MNIST Dataset.

Accuracy Score (Range 0-1) of the Ten Most Misclassified Letters on MNIST dataset							
KNN		Naïve Bayes		Logistic Regression		Random Forest	
M	0.978339	W	0.051948	X	0.843537	W	0.985836
W	0.978462	U	0.104348	V	0.861940	U	0.988701
V	0.985075	N	0.255014	R	0.909091	I	0.988701
X	0.986395	V	0.273616	K	0.917857	E	0.989865
S	0.986667	M	0.313846	S	0.926667	N	0.991228
U	0.989051	X	0.320225	U	0.930657	M	0.993311
E	0.991632	F	0.347222	I	0.931408	G	0.996815
I	0.992780	S	0.364943	W	0.932308	D	0.997135
N	0.993127	Y	0.398281	T	0.935374	S	0.997151
Y	0.993151	R	0.404145	F	0.935691	O	0.997175

Fig. 11. Most Misclassified Alphabets for the MNIST Dataset.

A. *MNIST Dataset*

In the first part of this research, we analyze the performance of the ML classifiers using the MNIST ASL detection dataset which contains images for 24 alphabets. The training set has 27,455 images, and test set contains 7,172 images. All images are 28x28 pixels which we convert to grayscale values (between 0-255). These images are cropped, which only contains hands, and have 15% brightness and contrast variation to get more accurate results in different illuminated environments. Table I shows four of the performance metrics for all the ML models that we build for the MNIST dataset. Figure 9 shows the confusion matrix for the same dataset to compare and see results of every character. Based on all the five performance metrics, Random Forest and KNN with k value arbitrarily chosen to be seven performs the best. Next, we show the top most classified and misclassified alphabets for the dataset in Fig. 10 and 11. As shown in Fig. 10, alphabets 'B', 'C' and 'P' are easily classified by all the four models, with alphabets 'H', 'O' and 'Q' next in line

TABLE II. PERFORMANCE METRICS FOR THE CLASSIFICATION MODELS FOR THE SELF-GENERATED DATASET.

Models	KNN	Naive Bayes	Logistic	Random Forest
Precision	0.74	0.18	0.25	0.39
Recall	0.71	0.16	0.26	0.25
F1-Score	0.71	0.13	0.25	0.25
Accuracy	0.71	0.16	0.26	0.25

with three of the four models correctly classifying the three alphabets. For identifying the most challenging alphabets, we observe 'W' appears in all four lists, while the letters 'S', 'X', 'M', 'U', and 'R' each appear in three of the four lists.

B. *Self-Generated Dataset*

In the second part of the research, we work on a self-generated dataset described in Section III. Similar to the MNIST dataset, we build four models namely, Naïve Bayes, KNN, Logistic Regression and Random Forest for building an ASL interpreter that can translate the sign language to English alphabets accurately. Table II shows the performance metrics: precision, recall, F1-score and overall accuracy. Additionally, we use confusion matrix shown in Fig. 12 to compare and see results of every alphabet. Based on the performance metrics, KNN performs the best among all the classifiers. The low performance of the other classifiers is expected as the quality of the images for the dataset we generate are captured using a basic webcam and very different from the images present in the MNIST dataset.

Additionally, we compute the per-class accuracies of the target alphabets we examine by extracting them from the confusion matrix generated by scikit-learn's confusion matrix function. We do this for every model we use so as to observe which alphabets are generally the most difficult to classify and which ones are generally the easiest. Fig. 13 and 14. The alphabets 'Q' and 'V' appear in each of the four lists, while the alphabets 'F', 'E', 'H', P', and 'K' each appear for three of the four ML methods of the lists, respectively. Fig. 14 shows the accuracies of the 10 most misclassified alphabets for each of our models. 'W' appears for all the four ML models, while the alphabets 'S', 'X',

'M', 'U', and 'R' each appear for three of the four models.

Next, we show the top most classified and misclassified alphabets for the dataset by computing the per-class accuracies of the target alphabets. We do this for every model we use to observe which alphabets are generally the most difficult to classify and which ones are generally the easiest. Tables I and II summarize the results obtained from scikit-learn's classification report for the classification models for the two datasets. The precision, recall, and f1-scores are the KNN classifier. From the diagonals, one can observe that KNN model is generally correct in classifying the twenty generally close to each other in terms of value, and they tend to mirror the overall accuracy scores. We create confusion matrices for each of the models with scikit learn and then feed them into Seaborn to create heatmap visualizations. Fig. 6 and 10 show the confusion matrix for four different alphabets for the datasets: 99% accuracy for the MNIST dataset and 71% accuracy for the new dataset we create for this research.

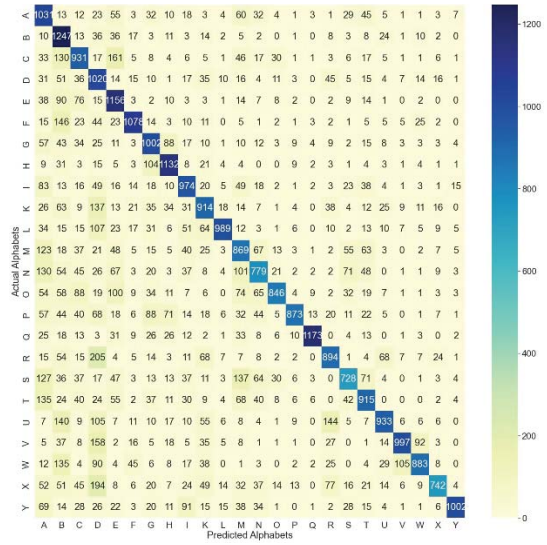

Fig. 12. Confusion Matrix for the test data for all the 24 alphabets A- Z represented as alphabets 0-23, for our self-generated dataset with the best performing classifier: KNN.)

Accuracy Score (Range 0-1) of the Ten Most Misclassified Letters			
KNN	Naive Bayes	Logistic	Random Forest
S 0.537546	W 0.011885	M 0.163447	A 0.104433
X 0.539598	U 0.015354	D 0.170654	Y 0.125574
N 0.560484	A 0.018868	I 0.173633	O 0.149265
O 0.585972	T 0.023697	C 0.181115	B 0.155885
M 0.632637	I 0.024888	W 0.185230	X 0.160213
T 0.637403	M 0.028892	S 0.186704	C 0.165026
U 0.641015	D 0.034283	L 0.186879	D 0.177469
P 0.649799	S 0.036062	T 0.194737	W 0.195253
W 0.675021	L 0.047368	R 0.195282	V 0.197989
R 0.680453	R 0.056461	X 0.197566	U 0.202435

Fig. 13. Most accurately classified alphabets for our self-generated dataset.

Accuracy Score (Range 0-1) of the Ten Most Accurately Classified Letters			
KNN	Naive Bayes	Logistic Regression	Random Forest
V 0.740784	N 0.098558	G 0.246962	E 0.243431
I 0.742777	F 0.129209	U 0.250000	S 0.246831
G 0.747244	E 0.179354	P 0.284921	R 0.283710
A 0.753612	P 0.197754	N 0.315239	K 0.305658
D 0.765601	Y 0.290503	E 0.319887	Q 0.334643
F 0.803991	B 0.300040	K 0.339976	M 0.345104
E 0.841859	H 0.315257	V 0.362354	F 0.375686
H 0.857481	K 0.545202	Y 0.373806	L 0.414188
Q 0.872213	Q 0.603206	H 0.396641	P 0.455951
B 0.896454	V 0.628571	Q 0.551474	N 0.496698

Fig. 14. Most misclassified errors for our self-generated dataset.

TABLE I. ACCURACY OF THE BEST MODEL: KNN FOR 10-FOLD CROSS VALIDATION APPROACH

MNIST Dataset	Self-Generated Dataset
99.42%	70.80%
99.53%	71.85%
99.49%	71.61%
99.53%	71.17%
99.60%	71.68%
99.82%	71.70%
99.49%	71.42%
99.67%	70.76%
99.71%	71.48%
99.49%	70.47%
Mean Accuracy = 99.57%	Mean Accuracy = 71.25%

In addition to a 70-30 train-test split we also apply 10fold cross validation for both the datasets. The results are shown in Table III. It can be observed that KNN achieves a minimum accuracy of 99.42% in each of the ten rounds of the cross validation with a mean accuracy of 99.57% for the MNIST dataset. Similarly for the self-generated dataset, we achieve above 70.40% accuracy in all cases with a mean accuracy of 71.25%.

VII. CONCLUSION AND FUTURE WORK

We build an ASL interpreter for each of the MNIST dataset and our-self created dataset (by using a low-profile laptop for taking images and creating the dataset) using KNN classifier. We achieve more than 99% accuracy for the existing dataset MNIST and more than 71% overall accuracy for the self-created dataset. The comparative lower accuracy for the latter dataset indicates that our self-created dataset needs additional preprocessing that may have been unexplored in this work or that there is a need for parameter tuning. For instance, the original size of our images is higher than 200x200 pixels, which are resized to 28x28 pixels, resulting in some information loss occurring during the resizing phase. Our camera specifications are another aspect that we do not consider while creating our dataset that most certainly has an impact as well.

For the future work, we would like to build a classifier that can work with non-static images, such that will take videos as inputs and be able to deal with dynamic signing with a goal of creating a general purpose ASL interpreter. Additionally we would like to consider hyperparameter tuning for our generated dataset along with detailed analysis of information loss while resizing the captured images with an original size of 200x200.

VIII. ACKNOWLEDGMENT

We would like to thank the Computer Science Department, and the ECS College Dean's office at California State University Fullerton for funding this work with the New Faculty Startup Fund.

REFERENCES

[1] Kushalnagar, Raja. "Deafness and hearing loss." Web Accessibility. Springer, London, 2019. 35-47.

[2] Goman, Adele M, and Frank R Lin. "Prevalence of Hearing Loss by Severity in the United States." American Journal of

[3] Public Health, American Public Health Association, Oct. 2016

[4] Waterfield, Sophia. "ASL Day 2019: Everything You Need To Know About American Sign Language." Newsweek, Newsweek, 15 Apr. 2019

[5] H. Kopka and P. W. Daly, A Guide to LATEX, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[6] American Society for Deaf Children https://deafchildren.org/, 2022.

[7] Triesch, J., & von der Malsburg, C. (2002). Classification of hand postures against complex backgrounds using elastic graph matching. Image and Vision Computing, 20(13-14), 937-943.

[8] Sadeddine, K., Chelali, F. Z., Djeradi, R., Djeradi, A., & Benabderrahmane, S. (2021). Recognition of user-dependent and independent static hand gestures: Application to sign language. Journal of Visual Communication and Image Representation, 79, 103193.

[9] Luzhnica, G., Simon, J., Lex, E., & Pammer, V. (2016, March). A sliding window approach to natural hand gesture recognition using a custom data glove. In 2016 IEEE Symposium on 3D User Interfaces (3DUI) (pp. 81-90). IEEE.

[10] Sahoo, J. P., Ari, S., & Ghosh, D. K. (2018). Hand gesture recognition using DWT and F-ratio based feature descriptor. IET Image Processing, 12(10), 1780-1787.

[11] Shin, J., Matsuoka, A., Hasan, M. A. M., & Srizon, A. Y. (2021). American sign language alphabet recognition by extracting feature from hand pose estimation. Sensors, 21(17), 5856.

[12] Sahana, T., Paul, S., Basu, S., & Mollah, A. F. (2020). Hand sign recognition from depth images with multi-scale density features for deaf mute persons. Procedia Computer Science, 167, 2043-2050.