CS3244 Project: Sign Language Detection: Performance Optimisation with MediaPipe Hand

Group 10: Nguyen Duc Danh (A0200689B) Guo Yichao (A0204731R) Li Yixuan (A0187289H) Yang Xinyi (A0177838J) Zhang Yilin(A0194570Y) Zheng Yuwei(A0188157R)

Abstract

The COVID-19 pandemic has deepened the communication barrier between the deaf community and those not equipped with sign language interpretation skills. Since mastering sign language can be challenging for humans, we introduced Machine Learning algorithms to uptake the task. In this paper, 4 different ML models are discussed for Sign Language finger spelling detection, including Logistic Regression, Random Forest, Bayes Classifier and Neural Network. A novel hand-tracking solution MediaPipe Hand is introduced to process noisy inputs such that the models trained can be better generalized to real-world sign language recognition.

1 Introduction

Singapore is estimated to have 500,000 people with hearing loss, which is approximately 9% of the total population(Deafness and Hearing Loss, n.d.). According to a report from the World Health Organisation (WHO), more than 5% of the world population is in need of assistance and rehabilitation methods, Sign language included, to address the hearing loss issue. Such a proportion is expected to rise to 8% by 2050. The universal finding can be extrapolated to the Singaporean context. Sign language is a very important and prevalent mode of communication the deaf community adopts to interact among themselves as well as with others outside the community. As the Singapore Association for the Deaf has noted, Sign language is a language natural to deaf people and is important to their self-identity (Singapore sign language course, n.d.).

The recent COVID-19 pandemic and the changes it has brought to our lifestyles have created many inconveniences to the deaf community. Their usual ways of communicating with the non-deaf are being disrupted. With the work/study-from-home policy, they can no longer depend on the interpreters' presence to help with the translation of Sign language. Writing and typing can be inefficient and therefore not timely enough in an online meeting. For physical meetups, masks cover their facial expressions, making sign language the only means of communication with the non-deaf. The deaf community is being inadvertently marginalized as they find it challenging to express their opinions in online real-time collaborations (Adam, 2021). Between struggling with masks and being left out of everything from virtual hang-outs to COVID-19 briefings, many deaf people suffered from isolation, which can be detrimental to their mental health. "Life has been much more isolating," says Poynter. She had already struggled to communicate with people who don't know ASL, "now everyone is having zoom parties and stuff, but deaf folks tend to be uninvited because we're not as thought about and the lack of accessibility (small cameras, laggy Internet, etc) make it extremely difficult to take part in any of it." (Sanchez, 2021)

Machine learning(ML) can provide an effective way to tackle the above-mentioned problem. Sign language is not a skill prevalently equipped by the public and the learning curve to master the skill is rather steep for humans, requiring much time and effort, practise, and memorization. However, with the current development in computer vision and machine learning, recognising hand gestures and their recurring patterns is no longer an impossible task (Data-flair, n.d.). With large data input to train the models, ML can provide efficient solutions to sign language detection and translation.

In light of this struggle the deaf community faces, we wish to build a hand gesture recognition application that can perform sign language translations for deaf people so that they can better voice out their opinions to people who do not know sign language. In our research, we attempt a diverse range of models, including Logistic Regression, Random forest, Bayesian Model and Neural Network. We hope our app can improve their disadvantaged position in a COVID-19 setting so that they can communicate more smoothly and feel more included in this difficult time.

2 Dataset

As a significant component of Singapore sign language follows the American sign language(ASL) standard, we make use of 2 ASL Kaggle datasets in our project to train and evaluate our models: the first dataset, denoted as **OG** consisting of a train set **OG**_train and a test set **OG**_test, is contributed by Kaggle user Akash; the second dataset, denoted by **ADD** is contributed by Kaggle user Dan Rasband. Both datasets are standard ASL datasets with images representing the 26 alphabets and "SPACE", "DELETE" and "NOTHING" from ASL (in total 29 classes).

In our project, we first select hyperparameters and train models with dataset **OG**_train. **OG**_train contains 78000 images, each of size $200 \times 200 \times 3$. **OG**_test is used for model comparison and selection. It consists of 28 images with a dimension of $200 \times 200 \times 3$ as well. However, due to the limited size of **OG**_test, **ADD** is used for a further evaluation of the models' performance. This is also to boost the variety of our test set to examine how well our models could generalise for real-world applications. **ADD** consists of 870 images, each with a resolution of 200×200. A detailed dataset preprocessing procedure is explained in the later section.

3 Methodology

The problem we seek to solve can be formulated as a supervised, classification problem. Hence, we try out 4 common classification algorithms that are applicable to this scenario and compare their performances (Shetty, n.d.):

- Logistic Regression (LR)
- Random Forest (RF)
- Bayes Classifier (BC)
- Neural Networks (NN)

There are 2 Phases of our experiments. Phase I trains models that directly recognize and classify the images, but unfortunately perform poorly when used for **ADD** image classification. With optimisation strategy offered by MediaPipe, Phase II trains models that utilise the coordinates of the important points on the hand providing gestural information.

3.1 Phase I

In Phase I, the gesture recognition algorithm was built using **OG**_train. The current dimension $200 \times 200 \times 3$ is too large for the ML models, so dimension reduction is performed to decrease computational cost and minimize overfitting caused by insignificant contributions to classification (Brownlee, 2020). For LR, RF, and BC, all the image datasets are converted to grey-scaled using $cv2.cvtColor$ and then reshaped to a 28×28 matrix. Further, the matrix is flattened to a one-dimensional array as the models accept one-dimensional input. For NN, the image is converted to $32 \times 32 \times 3$ to satisfy the requirement of the pre-trained model. RGB color is retained as the model could work well with colored images (i.e, 3-dimensional inputs).

Hyperparameter tuning is then carried out on preprocessed **OG**_train, via 5-fold cross-validation. A model selection and comparison is done with the preprocessed **OG**_test and **ADD**.

However, **OG**_train has an intrinsic limitation of sharing a similar background which has relatively little noise. As such, the accuracy obtained from Phase I models could be significantly lowered should they be used to predict gestures taken in real-life settings, which is why Phase II is introduced.

3.2 Phase II

In Phase II, we deploy a hand-tracking ML application MediaPipe to address the above-mentioned issue. MediaPipe Hand identifies and labels 21 keys points on the wrist, the palm, the knuckles, and the fingertips to assist hand gesture recognition. With only coordinates of the key points remaining, the background noise such as lighting and hand positioning is significantly reduced without losing the critical information about the gestures. Input in Phase 2 takes

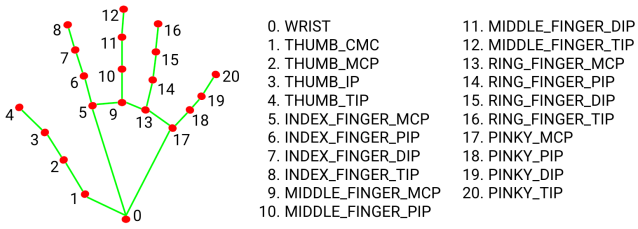

Figure 1: 21 Hand Landmarks Identified by MediaPipe

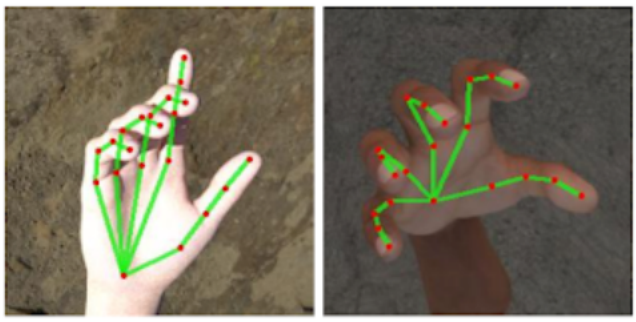

Figure 2: Hand Images with MediaPipe Annotation

the format of 21×3 matrix per image, with 21 points located by MediaPipe and their corresponding $x, y, andz$ coordinate values, and with z representing the depth of the labeled point to the camera. For each landmark detected, the x and y value is normalized to [0.0 - 1.0] with respect to the image widths and heights. Left and right-handedness are automatically corrected in the identification and labeling of the 21 key points. The selected ML algorithms retrain their models based on this new set of input.

3.3 Hyperparameter Tuning

We adopt a stratified k-fold-cross-validation(k-fold CV) method to assess the accuracy of our models and perform hyperparameter tuning. CV is a common statistical method to leave out a validation dataset from the training set to evaluate the accuracy of the prediction (Brownlee, n.d.). The training set is split into k parts in a stratified manner to ensure a similar proportion of different outcomes in the subsets, with one part reserved as a validation set and the $k-1$ parts as the training data. Hyperparameter tuning is performed on different combinations of the $k-1$ parts with the remaining one part as validation set to test out the performance of the model. The choice of k for our experiment is 5, so as to ensure the train-test ratio of 80% versus 20% (James, Witten, Hastie & Tibshirani, 2013).

3.4 Test Set OG_test Accuracy

OG_test is used to calculate the accuracy of the different algorithms when given unseen. It is not used in any of the model training procedures to prevent data leakage and over-optimistic estimate from the model.

3.5 ADD Further evaluation

To validate the performance of our model in real-world scenarios, we introduced an additional data set **ADD** of an image dataset of 870 images, with each image having a dimension of $200 \times 200 \times 3$. These images have a diverse background, and the accuracy of this test set implies the generalization ability of our models. Details of the experiments preparation and model selection processes are detailed in the following sections.

4 Results

4.1 Logistic Regression (LR)

The first model we attempted is a generalised Logistic Regression(LR) model. A softmax function is used to predict the probability of x_i belonging to a certain class k:

$$P(y^{(i)} = k|x^{(i)}; \theta) = \frac{exp(\theta^{(k)T} x^{(i)})}{\sum_{K}^{j=1} exp(\theta^{(j)T} x^{(i)})}$$

A classification of x_i is done via majority rule. The parameter θ is consistently updated to improve the performance of LR model. This is equivalent to minimising the cross-entropy loss function, which calculates the error between the actual and predicted classes.

Phase I In Phase I, **OG**_train dataset is used to train LR model, consistently updating θ of size 784 x 30 in softmax.

Mini-batch gradient descent is adopted for a faster updating process, due to the high dimension of inputs and parameters. Several hyperparameters involved include $batch\_size$, *number_of_training_steps* (for gradient descent), and *learning_rate*. A 5-fold CV is adopted for model training and selection of hyperparameters.

As shown in Figure 3, a *batch_size* of 1000 is ideal: with *batch_size* of 1000, LR achieves a relatively high average validation accuracy. A *number_of_training_steps* of 18000 is used because validation accuracy starts to converge afterwards. For Mini-batch GD, a *learning_rate* is carefully chosen to prevent slow and wasteful updating processes(for small learning rate) or failure of convergence(for large learning rate). *Learning_rate* of 0.05 is chosen for best validation performance. With the selected hyperparameters, we retrain the LR model on **OG**_train. It takes 87 seconds to train this LR model. The test accuracy is 14.30% on **OG**_train, but only 2.80% accuracy in **ADD** set, indicating a low generalisation capability.

Phase II In Phase II, LR takes in the flattened arrays of the coordinates array with a length of 21×3 as inputs. Based on the validation accuracy, it seems that LR has a better performance with a *batch_size* = 200, *step_size* = 20000 and *learning_rate* = 0.01 (Figure 4). It takes 65 seconds to train this LR model. Accuracy is improved to 77.30% for **OG**_train and 75.20% for **ADD** set, showing a high fidelity of the model in generalized situations.

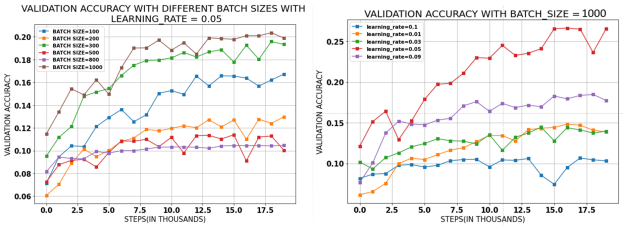

Figure 3: Hyperparamter Tuning for LR in Phase I

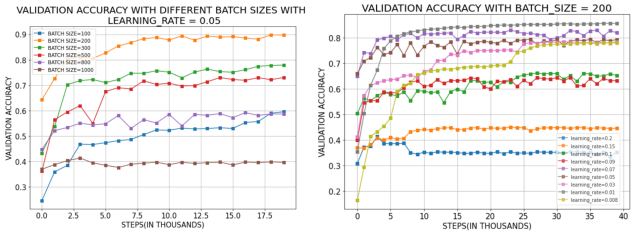

Figure 4: Hyperparamter Tuning for LR in Phase II

4.2 Random Forest (RF)

The second model attempted is generalized Random Forest, a process of randomly generating different decision trees, and treating them as indifferent judges to achieve a majority voting. Based on statistics, these majority voting usually arrives at an accurate and stable prediction. The prediction on unseen examples is made by averaging predictions on all individual trees x'.

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x')$$

The parameters to be considered by Random Forest:

- **max_features**: the maximum number of features allowed in each individual tree, default = n_features a.k.a. number of variables to consider, 'sqrt' = $\sqrt{n\_features}$

Table 1: Accuracy Score for parameter max_features

	Image (Phase I)		Image (Phase II)	
value	Test_OG	Test_ADD	Test_OG	Test_ADD
n_features	40.63	2.87	59.09	21.31
sqrt	31.81	2.30	51.08	10.74
10	28.47	2.41	50.90	9.20

- **n_estimators**: the number of trees built before majority voting, default = 100

Table 2: Accuracy Score for parameter n_estimators

	Image (Phase I)		Image (Phase II)	
value	Test_OG	Test_ADD	Test_OG	Test_ADD
1	85.14	3.56	95.45	53.14
10	55.24	4.02	81.82	22.92
100	40.63	2.87	59.09	21.31

- **min_samples_leaf**: the minimum number of samples allowed at a leaf node in the decision tree, default = 1

Table 3: Accuracy Score for parameter min_samples_leaf

	Image (Phase I)		Image (Phase II)	
value	Test_OG	Test_ADD	Test_OG	Test_ADD
1	40.63	2.87	59.09	21.31
3	37.01	3.33	45.45	20.93
10	31.88	3.93	40.91	19.41

Phase I It takes around 1 hour to train the model of all hyperparameters as default values, achieving an validation accuracy of 40.62% on Test_OG without tuning.

Phase II It takes around half an hour to train the model with all hyperparameters as default values, achieving an validation accuracy of 59.09% Test_OG without tuning.

Effect of tuning hyperparameters :

The effect of tuning hyperparameters are similar on both Phase 1 and Phase 2. Tuning of *n_estimators* can significantly increase the accuracy. By minimizing *n_estimators*=1, we achieve an accuracy of 85.14% in Phase 1 and 95.45% in Phase 2. The effect of tuning other hyperparameters is mild. Still, minimizing *min_samples_leaf* and maximizing *max_features* leads to better accuracy score. This is reasonable because the more detailed and meticulous our decision trees are (considering more features and splitting more branches), the more likely our predictions will be, despite taking longer time to train.

As for the best accuracy possible reason is that the coordinates themselves are interdependent, and extensive clustering of trees disrupts such dependency, leading to poor performance. The best accuracy of the validation sets reported is 85.14% and 95.45% after tuning, which are the cases of only using only one decision tree.

Effect of ADD dataset : The performance of both the Phase I and the Phase II on the Test_ADD dropped significantly compared to the performance on its Test_OG (from 85.14 to 3.56 in Phase I and from 95.45 to 53.14 in Phase II). We can see that the performance of model built during Phase I in predicting the Test_OG(with non-monochromatic background) is no better than random guesses whereas Phase II model still maintains certain level of accuracy. It suggests that the randomness in the background in grayscale encoding has much more significant impact than the randomness in background in coordinate data.

Comparing Phase I and Phase II :

The overall performance of RandomForestRegressor in Phase II is better than in Phase I.

In Phase II. the coordinates inputs are multiplied by a factor of 500 to approximate an integer as required by a decision tree. As the size of attributes for each input is reduced from 28x28(784) to 21x3(63) compared with Phase 1, the training takes less time. The accuracy_score is also universally better.

4.3 Bayes Classifier (BC)

The third model attempted is the Naive Bayes model, a classification technique based on the Bayes' Theorem with an assumption that a particular feature in a class is unrelated to the presence of any other feature. The key hyperparameter considered for Bayesian Inference is the Naive Bayes models used. There are 4 possible choices: the first one is the Gaussian Naive Bayes classifier (GNB), which assumes that the features are following a normal distribution; the second one is Bernoulli Naive Bayes (BNB) classifier, which is useful if we use binary feature vectors; the third one is Multinomial Naive Bayes (MNB) classifier, which is used when the features follow a multinomial distribution; the fourth one is the Complement Naive Bayes (CNB) classifier, which implements the complement naive Bayes algorithm and is suitable for imbalanced data sets.

Table 4: Accuracy for the Bayes Classifiers

Model used	GNB	BNB	MNB	CNB
Phase I Accuracy	20.35	3.06	15.60	14.55
Phase II Accuracy	44.36	15.26	45.97	39.20

Phase I Gaussian Naive Bayes performs the best in Phase I among the four Naive Bayes classifiers after CV, with an accuracy of 20.35% (Table 4).

Phase II In Phase II, in order to avoid negative values occurring in the training data, MinMaxScaler is used to preprocess the coordinates. Gaussian Naive Bayes classifier is again proved to be the classifier with the highest accuracy, compared to Phase I, with an accuracy of 44.36%. The results obtained indicate an obvious improvement of accuracy from Phase I to Phase II, which is evidence of the effectiveness of dimension reduction using MediaPipe.

4.4 Neural Network (NN)

The last model that we attempted is Neural Network (NN). Phase I of NN makes use of a pre-trained model Vgg16, a vision model developed by the Vision Geometry Group from oxford, to classify sign language images given. The pixel value is normalized by 255 to a value between 0 and 1. The labels are one-hot encoded.

Phase I We construct a Convolutional Neural Network (CNN) via layer addition. Firstly, we added the a layer to flatten the input image (without preprocessing). A dropout layer was added to help prevent over-fitting for the Neural

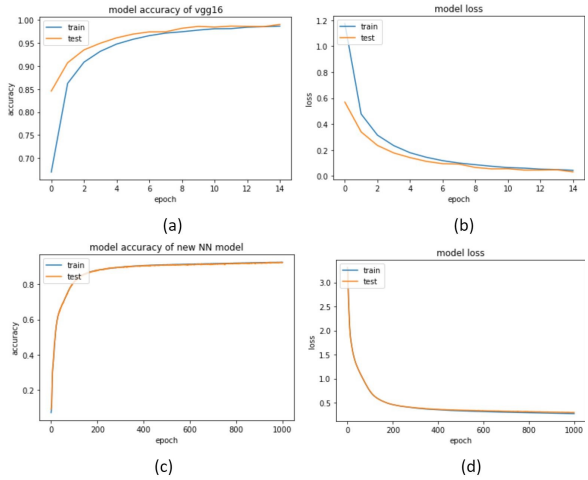

Figure 5: Hyperparameter Tuning for Neural Network

Network. The final layer outputs the classification in the correct format of the label.

A few hyperparameters are accounted for, including the *batch_size*, *number_of_epochs*, and the *dropout_rate* in the dropout layer. With 5-fold CV, the model with the best performance has *batch_size* 64, a *number_of_epochs* 14, and a *dropout_rate* of 0.1 for Phase I (Figure 5a & 5b). We obtained an accuracy of 99.052% for **OG** test set images and 100% for **OG** training set images. However, a further test on the additional dataset Kaggle **ADD** yields a poor accuracy of 17.63%. This shows that our model built based on image input has a low generalisation capability when predicting real-world data.

Phase II For Phase II, as the input format has changed from images to coordinates, we construct an ANN with 2 hidden layers and 1 output layer. The best set of hyperparameters selected after CV is 100 for the *number_of_epochs* and 100 for *batch_size* (Figure 5c & 5d). The accuracy on **OG**_test coordinates is 92.83% while the accuracy on **OG** training coordinates is 94.46%. When tested again with the unseen Kaggle dataset **ADD**, the model yields an accuracy of 84.511%. From Phase I to Phase II, the model's performance on unseen data has improved significantly. The coordinate transformation on images has successfully boosted the model's generalization ability.

Moreover, the model takes significantly less time for training in Phase II (15 minutes for 1000 epochs) compared to Phase I (2.5 hours for 15 epochs). The dimension reduction method by MediaPipe is effective in reducing computation time.

A conclusion of all the accuracy of different models across the two Phases are summarized in Table 5.

Table 5: Summary of Accuracy

	Image (Phase I)		Image (Phase II)	
	Test_OG	Test_ADD	Test_OG	Test_ADD
LR	14.30	2.80	77.30	75.20
RF	85.24	3.50	95.45	50.30
BC	20.35	2.91	44.36	46.70
NN	99.05	17.63	92.83	84.51

5 Discussion

5.1 Models and Their Accuracy

Overall, model accuracy show good improvement from Phase I to Phase II when the models are applied to the additional test set, suggesting that coordinates help simplify the model, reduce the time needed for training and improve the generalizability and performance of our models. The disparity of accuracy between the **OG**_test set and **ADD** test set indicates the limited ability of the model built on image inputs to process noisy input, providing justification for our attempt to use MediaPipe to reduce the additional interference from the background. In general, Neural Network is the best performing model for both image and coordinate input, with high fidelity of generalizability. Although Random Forest seems to demonstrate better performance on **OG**_test, the accuracy drops significantly when applied to **ADD** set and is outperformed by the Logistic Regression model.

Convolutional Neural Network is traditionally used for computer vision and image recognition. Although previous work has reported a 95.54% accuracy for the same Vgg16 model, we suspect that the high accuracy is caused by the similar background noise for the input images as the data was augmented around 6 fold from an original set of 2524 images(Masood, Thuwal & Srivastava, 2018). Our results (17.65%) suggest that in the real-world application of sign language detection, existing CNN models may not suffice for a noisy input. Additional methods such as adopting landmark recognition MediaPipe help increase the ML models' ability to predict real-world unseen data drastically(84.51%).

Previous studies have also explored the use of the Random Forest for ASL fingerspelling detection and suggested that a combination of appearance and depth of images gives the best performance of 75.00% in real-time detection (Pugeault & Bowden,2011). Our study can be considered as an extension as not only the depth (reflected by the z coordinates) but also the key landmarks on a hand are taken into consideration. Nevertheless, further optimization is needed as real-time detection needs to deal with motion pictures.

Logistic Regression and Bayesian models are not conventionally used for sign language detection, as they are considered more "naive" models. Still, we observe a significant increase when the input is shifted from image to coordinates. For future research, an attempt to layer different models may yield good results, especially if additional information can be collected, for example, the prior distribution of different alphabets in ASL.

5.2 Discussion of Methodology

There are several nuances in our methodology to be discussed here. Firstly, we utilize accuracy instead of the conventional f1 score for performance measurement of classification problems. After inspection, we observed that the ASL

dataset (**OG**_train, **OG**_test and **ADD**) is relatively balanced, with a similar frequency for each alphabet label, justifying the use of accuracy. Secondly, NN and LR both use mini-batch gradient descent. However, the batches are created and chosen randomly and separately for the two models, which may cause some variations in the results. What's more, data preprocessing also interferes with model performance. For example, the dimension reduction in Phase I from (200, 200, 3) to (784, 1) or (32, 32, 3) is achieved by reducing the resolution of images. The choice of dimensions is chosen in an arbitrary manner. A different choice of dimensions may yield slightly different results, which is not explored in the current report. Moreover, in this report, hyperparameter tuning is done by 5-fold CV. Other methods such as Grid Search and Random Search could be further explored for a different model performance.

5.3 Further Improvement

Still, our model can be improved by addressing the following issues. Firstly, the use of MediaPipe to generate coordinates may not be the best approach. As an ML algorithm itself, MediaPipe is not guaranteed to have a 100% accuracy in its coordinates identifications. Images with poor lighting and overlapping hand landmarks may not be effectively recognized. Hence, the actual accuracy of the model applied in real-life scenarios would likely be compromised as it is subjected to the performance of MediaPipe. Also, as MediaPipe performs hand gesture identification based on the identification of the palm, it may demonstrate a decrease in accuracy when only the side view of the hand is presented. Besides, there are certain ASL alphabets that are expressed in terms of motion instead of a static gesture. Our model is currently unable to capture and identify motion signals and is only applicable to frames of images taken from videos. Finally, finger spelling is only a small subset of sign languages. Future research may need to address phrase detection in ASL which requires more extensive classification based on the ASL dictionary.

6 Conclusion

In conclusion, 4 ML models were performed and compared in sign language detection. Our results have provided us with the following key insight. Firstly, when an image is used as input, the model tends to be heavily biased on the training data. **OG** test has a significantly higher accuracy as compared to **ADD** test, the additional test with various background noise used to represent the real-world situations. Secondly, using landmark coordinates seems to demonstrate high performance in reducing computational run time, model complexity and most importantly increasing accuracy, especially for unseen real-world data. Nevertheless, the current research is a preliminary attempt for the detection of ASL in static images. Future research should continue to address the problem of input simplification for real-time and motion-based sign language detection.

References

[1] Adam, R. (2021, March 15). Sign languages and deaf people during COVID-19: How you can help in the classroom. Retrieved from https://www.cambridge.org/elt/blog/ 2020/09/22/sign-languages-and-deaf-people-during-covid-19-how-you-can-help-in-the-classroom/

[2] Akash. (2018, April 22). Asl alphabet. Retrieved April 16, 2021, from https://www.kaggle.com/grassknoted/asl-alphabet

[3] ANNALS Academy of MEDICINE Singapore - continuing professional development. (n.d.). Retrieved from https://www.annals.edu.sg/cpdMay05.html

[4] Brownlee, J. (2020, June 30). Introduction to dimensionality reduction for machine learning. Retrieved from https://machinelearningmastery.com/dimensionality-reduction-for-machine-learning/

[5] Brownlee, J. (2020, August 02). A gentle introduction To k-fold cross-validation. Retrieved from https://machinelearningmastery.com/k-fold-cross-validation/: :text=Cross%2Dvalidation%20is%20primarily%20used, the%20training%20of%20the%20model.

[6] Data-flair. (2021, March 08). Advantages and disadvantages of machine learning language. Retrieved from https://data-flair.training/blogs/advantages-and-disadvantages-of-machine-learning/

[7] Deafness and hearing loss. (n.d.). Retrieved from https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

[8] James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. An introduction to statistical learning (Vol. 112, p. 18). New York: springer.

[9] Masood, S., Thuwal, H. C., & Srivastava, A. (2018). American sign language character recognition using convolution neural network. In Smart Computing and Informatics (pp. 403-412). Springer, Singapore.

[10] Pugeault, N., & Bowden, R. (2011, November). Spelling it out: Real-time ASL fingerspelling recognition. In 2011 IEEE International conference on computer vision workshops (ICCV workshops) (pp. 1114-1119). IEEE.

[11] Rasband, D. (2018, August 01). ASL alphabet Test. Retrieved April 16, 2021, from https://www.kaggle.com/danrasband/asl-alphabet-test

[12] Sanchez, K. (2021, January 29). Deaf people face unique challenges as pandemic drags on. Retrieved from https://www.theverge.com/22254591/deaf-communication-tech-access-coronavirus-isolation

[13] Singapore sign language course. (n.d.). Retrieved from https://sadeaf.org.sg/sgsl-course/

[14] Singapore sign language course. (n.d.). Retrieved April 16, 2021, from https://sadeaf.org.sg/sgsl-course/

[15] Shetty, B. (n.d.). An in-depth guide to supervised machine learning classification. Retrieved from https://builtin.com/data-science/supervised-machine-learning-classification

7 Appendix

7.1 Role for each member

Data Pre-processing Image: Li Yixuan
MediaPipe: Yang Xinyi, Zheng Yuwei

Models LR: Li Yixuan; RF: Guo Yichao; BC: Zhang Yilin; NN: Nguyen Duc Danh

Final Write Up Yang Xinyi, Zheng Yuwei