50.039 Big Project Report

Word-Level American Sign Language (WLASL)
Sign Language Recognition
Group 6

Suhas Sahu (1003370)
Sidharth Praveen (1003647)
Noorbakht Khan (1003827)
Richard Pung Tuck Wei (1003523)
Ong Li-Chang (1003328)

Table of Content

Background and Objective

As part of our Big Project of 50.039 Deep Learning, our team decided to tackle isolated sign language recognition from signer-independent videos involving a large number of sign categories.

Dataset download link

https://drive.google.com/drive/folders/1ekKwXXHfoxTkyysuWKwvNRQc1Le73og9?usp=sharing
(Please email Suhas Sahu at suhas_sahu@mymail.sutd.edu.sg if the folder has problems)

Dataset and Exploration

For the development of our project, we used the WLASL Dataset, which is the largest video dataset for Word-Level American Sign Language (ASL) recognition. It features 2000 common different words in ASL. However, due to limited computational power and complexity, we used the WLASL100 subset as our dataset, which contains 100 words(glosses) instead of 2000. Each word has training copies ranging from 18 to 40 in the dataset. The mapping between videos and words are stored in a JSON file, which is passed into the Dataset object. Additionally, this JSON file also contains the bounding box details as well as the relevant frames that we need to consider for each word.

JSON Attributes:

1. `gloss`: str, data file is structured/categorised based on sign gloss, or namely, labels.
2. `bbox`: [int], bounding box detected using YOLOv3 of (xmin, ymin, xmax, ymax) convention. Following OpenCV convention, (0, 0) is the up-left corner.
3. `fps`: int, frame rate (=25) used to decode the video as in the paper.
4. `frame_start`: int, the starting frame of the gloss in the video (decoding with FPS=25), indexed from 1.
5. `frame_end`: int, the ending frame of the gloss in the video (decoding with FPS=25). -1 indicates the gloss ends at the last frame of the video.
6. `instance_id`: int, id of the instance in the same class/gloss.
7. `source`: str, a string identifier for the source site.
8. `split`: str, indicates sample belongs to which subset(train/test)
9. `url`: str, used for video downloading.
10. `video_id`: str, a unique video identifier.

Data Augmentation

Converting Images To Grayscale

For the purposes of classifying sign language, colour is not necessary to determine shown gestures. To that end we removed colour from our input by applying a grayscale technique. This will allow us to remove complexity from our calculation as grayscale images are more computationally efficient.

Data Normalisation

We applied data normalisation to the dataset which allowed us to alter the values of the numeric columns of the image tensor dataset to a common scale without distorting differences in the range of values. This can give better performance as the coefficients of linear models are made more interpretable.

Cropping

Random crop is a data augmentation technique where a random subset of an original image is created. This may help models to generalize better because the object(s) of interest we want our models to learn are not always fully visible in the image or the same scale in our training data. However, we realised that some hand signs are performed below chest level (Figure 1) while some hand signs are performed above chest level (Figure 2). Random cropping may result in the images to miss out key information of the hand signs. Thus, we left out random cropping after qualitatively evaluating it.

Figure 1: ASL below chest level

Figure 2: ASL above chest level

Model Architecture

For our model architecture, we implemented a Graph Convolutional Network (GCN) consisting of stacked blocks of GCN and BatchNorm layers.

GCN Layer

The base GCN layer uses the conventional spectral propagation rule. Two calculations will have to be made, the first a matrix multiplication between weights of the current layer, W, and the input. This is similar to the output of a linear fully-classified layer. This produces the support equation. This support equation will then be matrix multiplied with the embedding of the node in the adjacent layer and this will be our output. We attach a bias to this output and this will be our finalised output for our basic GCN layer (Kipf & Welling, 2016) (Stern, 2020).

GC Block

Using the base layer, we built a reusable GCN and BatchNorm layer block. The batch normalisation helps to standardise input to layer for each minibatch which helps to stabilise the learning process.

GC Multi Block

For the final GCN model, we will stack the GC blocks. The final layer of the model will be a fully-connected linear layer to allow for classification. Dropout is also implemented to randomly remove some neurons in order to prevent over-fitting. In addition to that, residual links are used to assist in gradient flow.

For a more detailed explanation, please view the included iPython notebook, Notebook_Submission.ipynb, which has comments explaining the relevant code sections and package dependencies.

Model Parameters

Epochs	Dropout	Batch Size	Loss Function	Optimiser
50	0.3	64	Cross Entropy Loss	Adam (LR = 0.001, EPS = 1e-3, Decay = 0)

Epochs

A major consideration for training the model was time taken. From experimentation, 50 epochs gave the best time to performance ratio.

Dropout

As a precautionary measure to prevent the model from overfitting, dropout was implemented. This was especially relevant for our model given that we were analysing gestures that may be placed differently in each sample. From experimentation, 0.3 gave better results over the default 0.2.

Batch Size

To take full advantage of the graphics processing unit (GPU), the number of batch sizes should be a power of 2 (Kandel and Casteli, 2020). Given that batch size can have a significant impact on the performance we experimented with batch sizes ranging from 32, 64 and 128. The batch size of 64 was ultimately chosen as it balanced both computational time and model performance.

Loss Function

Cross-entropy loss was chosen as this was a multi-class classification problem. This would allow the model to compute the most probable class for specific input.

Optimizer

Adam optimiser was used as it is the current default algorithm of choice and often works slightly better than RMSProp.

Performance and Results

Custom Model (w/o Aug)	Custom Model (w Aug)
19.77%	19.38%

Table 1: Comparison of performance between custom models run for 20 epochs

Model	Custom Model (w/o Aug)	Pose-GRU	Pose-TGCN	VGG-GRU	I3D
Top-1 Accuracy (%)	34.11	46.51	55.43	25.97	65.89

Table 2: Comparison of performance between models

Visualisation of Model's Performance

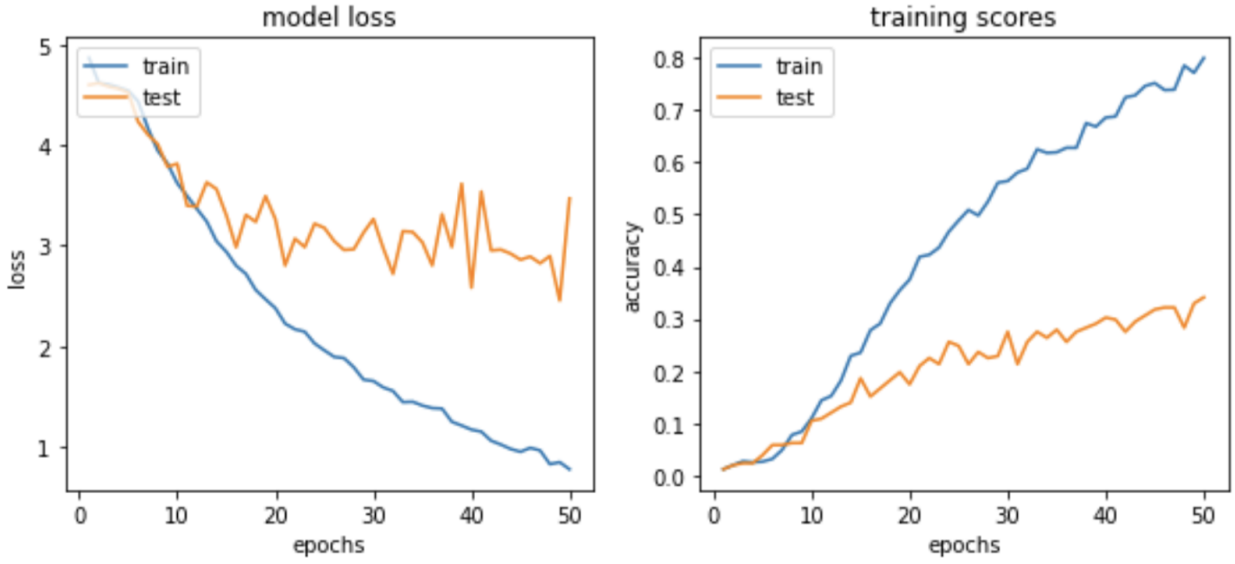

Figure 3: Model Loss and Training Scores - Train and Test

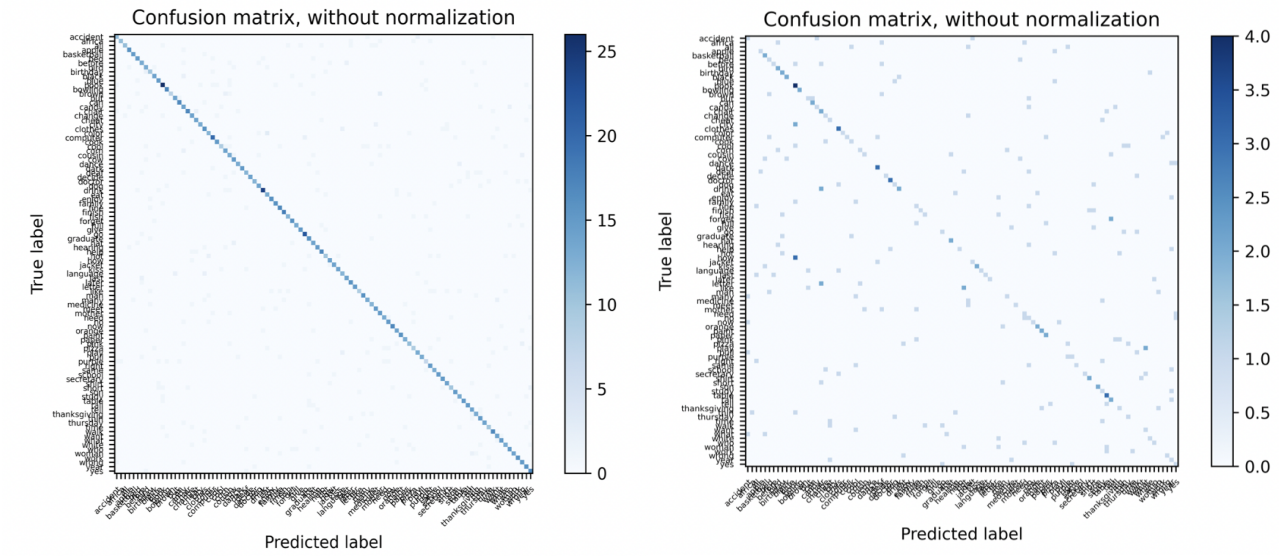

Figure 4: Confusion Matrix - Train and Test

A key consideration was whether data augmentation proved to be advantageous. As such, we ran each of the models for 20 epochs and compared the scores. In 20 epochs, the model was able to attain an accuracy of 19.77% without augmentation, however achieved an accuracy of ___ with augmentation. A possible reason could be that applying grayscale led to certain indicators being lost that may have been essential for the model to identify the sign language. Following the initial experimentation between the 2 models, we proceeded to continue running the model without any augmentation for 50 epochs.

As it can be observed from the plots in Figure 3, the training loss and accuracy do exhibit learning characteristics over the dataset. We do not observe any trend of overfitting, as the validation loss does decrease with slight oscillations. The validation accuracy also improves with time. Furthermore, the confusion matrix (Figure 4) on the validation set does show promising trends, as we can observe ample amounts of correct label predictions (the diagonal trend). We hypothesize that with more computational power and time, we would be able to try out different configurations of the model, and finetune the hyperparameters to a greater extent.

Model Recreation Steps (From Scratch)

To train the model:

Run python train.py (default parameters comprise of 50 epochs, 64 batch size and 0.001 learning rate)

Relevant files:

- asl100.ini - configuration file containing parameters for training, optimizer and parameters for our GCN model
- configs.py - file containing the
- dataloader.py - file containing the Dataset class and auxiliary functions for creating the dataset object
- model.py - file containing our GCN model
- train.py - file containing training and validation functions
- utils.py - file containing one-hot encoding functions and relevant functions for plotting curves and confusion matrix.

Loading Trained Model Weights

Please find saved model weights in the ‘weights’ folder. To load a model, run the first two cells of the submission notebook, and then run the last cell. The function takes in the path of any saved weight and loads the model. This then runs the model on evaluation mode and calls the validation function, which reproduces the accuracy we obtained.

Model Malfunction Examples

User Interface Demo

The user interface demo can be found on the following YouTube link: https://youtu.be/V1A4gUTmzjw. It contains an example of a successful prediction and an unsuccessful demonstration. The code used to generate this demonstration can be found on the github link provided earlier, 'test_w_ui.py' or is accessible from the following link: https://github.com/sidharth3/sign-language-recognition/blob/main/test_w_ui.py

The user interface demo used to test the model's performance is shown below. The steps can be broken down as follows: (1) Select the MP4 file containing the video that has to be translated (2) Wait for the model to run (3) Rate the prediction, if it was accurate for not. This serves as a means to collect extra data points that can be used for further training.

The number right below the 'Drag and drop file here' box provides the unique 'id' of the video that has been selected. This 'id' is then used to cross reference the existing database to check if the prediction is correct.

Malfunction 1

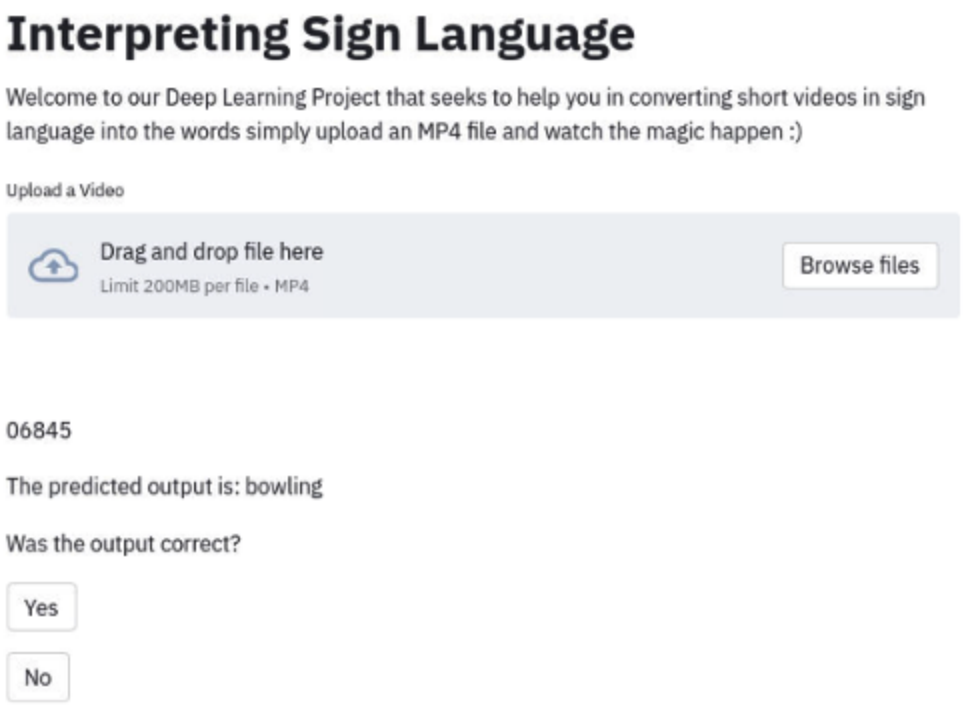

Figure 5: Model malfunction 1

The predicted output is “bowling” but the actual output is supposed to be “blue”. An investigation of this error showed the gesture for “bowling” was quite different from that of blue, as the figures below depict. This reflects the relative inaccuracy of the model that was trained.

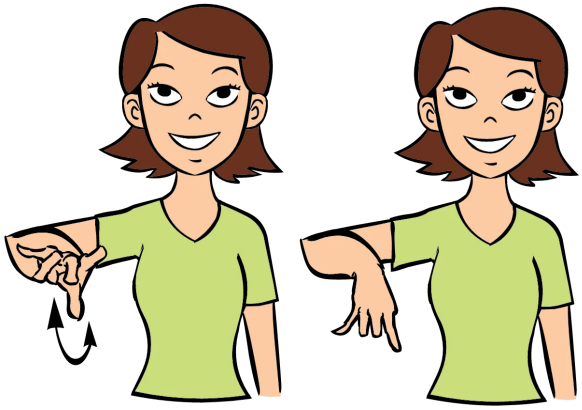

Figure 6: ASL representing “bowling”

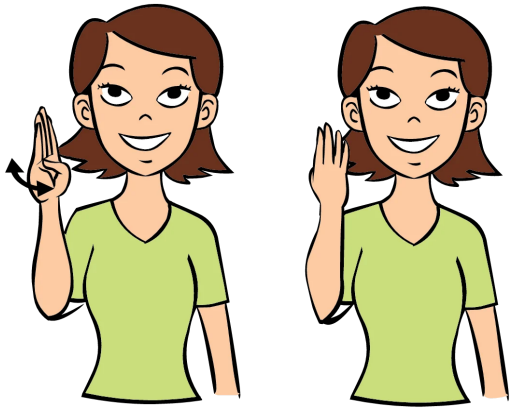

Figure 7: ASL representing “blue”

Malfunction 2

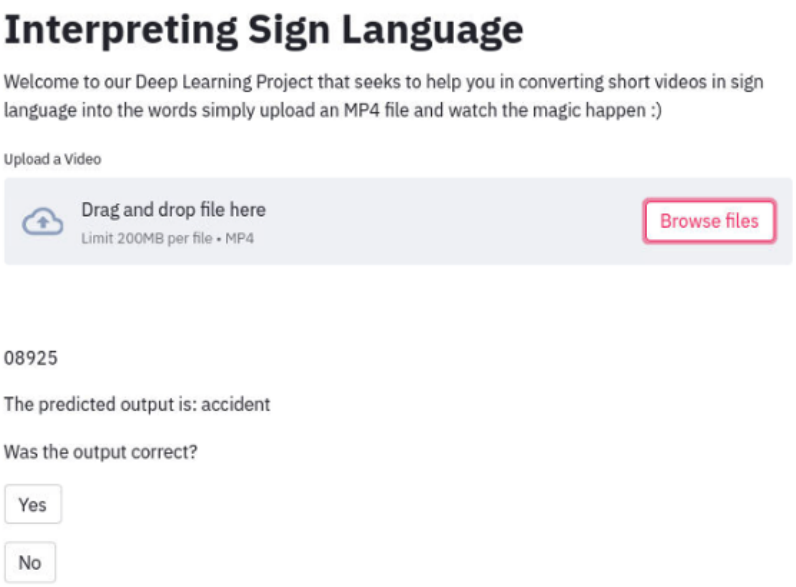

Figure 8: Model malfunction 2

The predicted output is "accident" but the correct label is "candy". The images below show that the model was able to capture coarse features such as the location of the hand, but not able to capture the hand gesture, which is a finer feature.

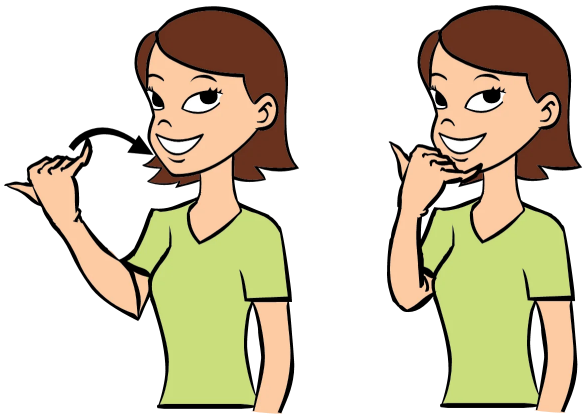

Figure 9: ASL representing "accident"

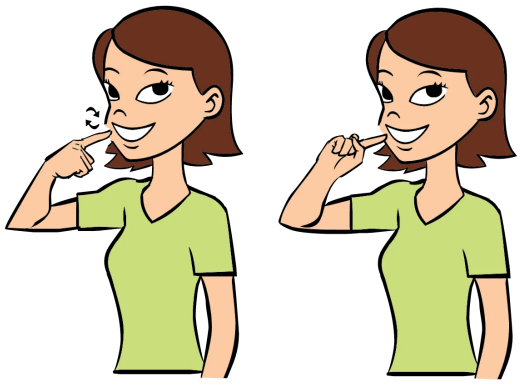

Figure 10: ASL representing "candy"

Group Member's Contributions

Team Member	Contributions
Suhas Sahu	Dataloader and UI
Richard Pung	Dataloader and Data Augmentation
Noorbakht Khan	Model and Report
Ong Li-Chang	Model and Report
Sidharth Praveen	Train Function and Integration

Table 3: Group members contributions

References

1. *Accident*. Baby Sign Language. (2021, April 27). https://www.babysignlanguage.com/dictionary/accident/.
2. *Blue*. Baby Sign Language. (2021, April 28). https://www.babysignlanguage.com/dictionary/blue/.
3. *Bowling*. Baby Sign Language. (2021, April 27). https://www.babysignlanguage.com/dictionary/bowling/.
4. *Candy*. Baby Sign Language. (2021, April 28). https://www.babysignlanguage.com/dictionary/candy/.
5. Kandel, I., & Castelli, M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, *6*(4), 312–315. https://doi.org/10.1016/j.icte.2020.04.010
6. Kipf, T. N., & Welling, M. (2016). *Semi-Supervised Classification with Graph Convolutional Networks*. ICLR. https://arxiv.org/abs/1609.02907
7. Stern, F. (2020, May 22). *Program a simple Graph Net in PyTorch - Towards Data Science*. Medium. https://towardsdatascience.com/program-a-simple-graph-net-in-pytorch-e00b500a642d