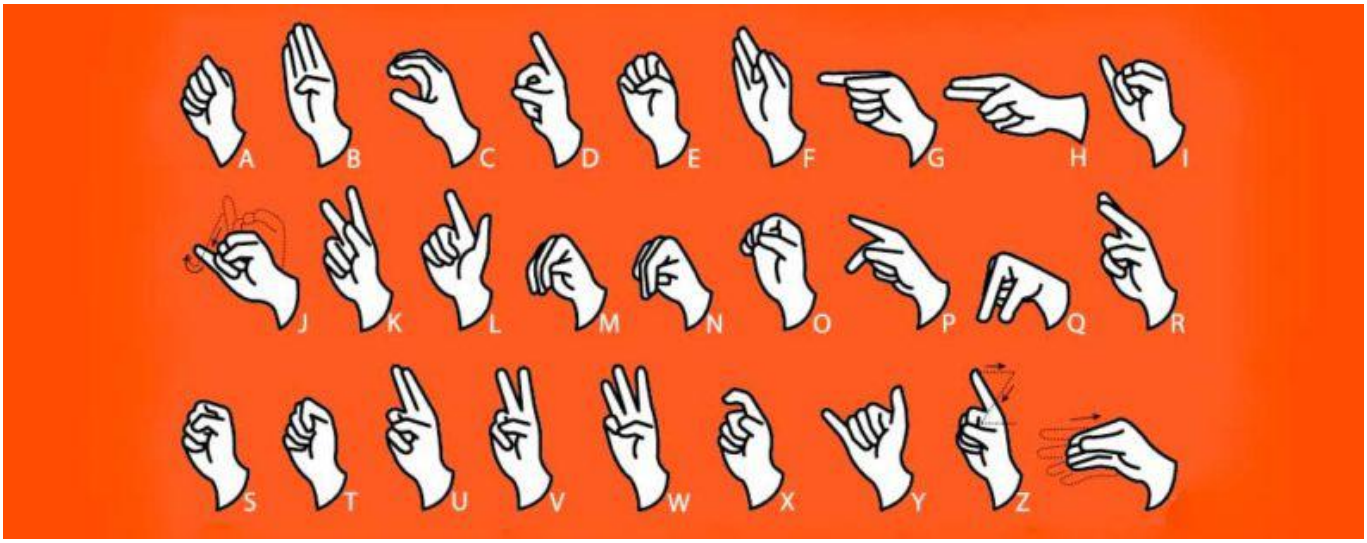


1 Introduction

The goal of this project was to build a neural network able to classify which letter of the American Sign Language (ASL) alphabet is being signed, given an image of a signing hand. This project is a first step towards building a possible sign language translator, which can take communications in sign language and translate them into written and oral language. Such a translator would greatly lower the barrier for many deaf and mute individuals to be able to better communicate with others in day to day interactions.

This goal is further motivated by the isolation that is felt within the deaf community. Loneliness and depression exists in higher rates among the deaf population, especially when they are immersed in a hearing world [1]. Large barriers that profoundly affect life quality stem from the communication disconnect between the deaf and the hearing. Some examples are information deprivation, limitation of social connections, and difficulty integrating in society [2].

Most research implementations for this task have used depth maps generated by depth camera and high resolution images. The objective of this project was to see if neural networks are able to classify signed ASL letters using simple images of hands taken with a personal device such as a laptop webcam. This is in alignment with the motivation as this would make a future implementation of a real time ASL-to-oral/written language translator practical in an everyday situation.

2 Description of Overall Software Structure

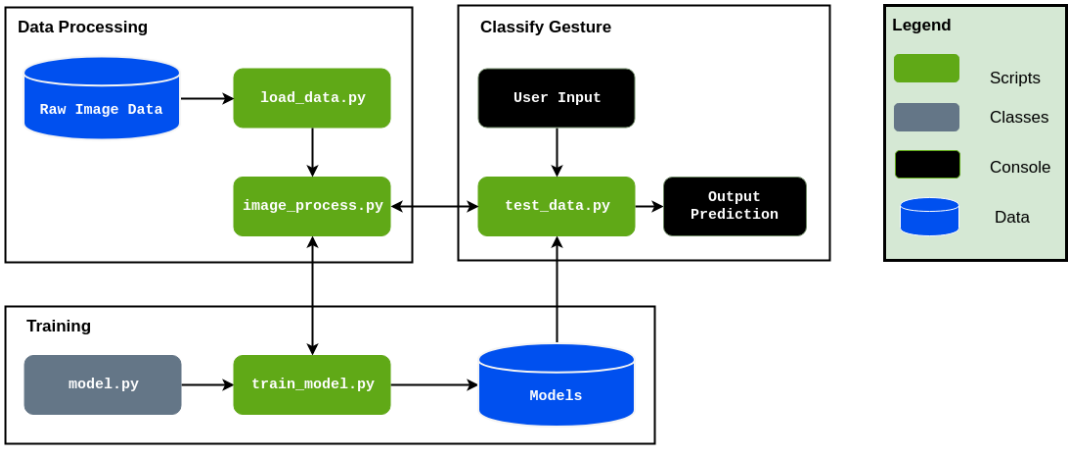


Figure 1: Block Diagram of Software

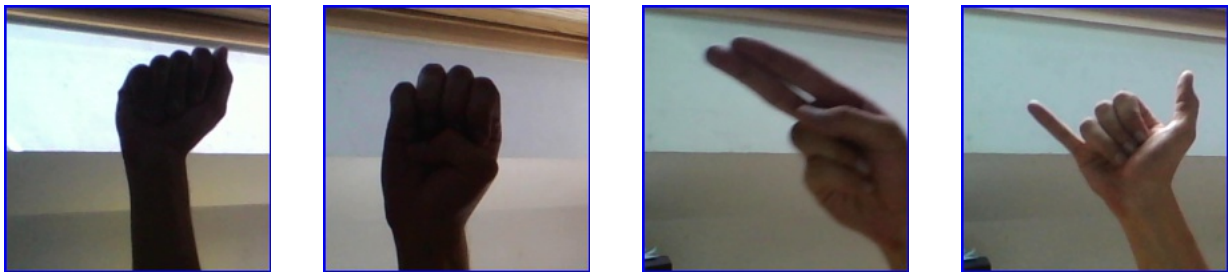
As shown in Figure 1, the project will be structured into 3 distinct functional blocks, **Data Processing**, **Training**, **Classify Gesture**. The block diagram is simplified in detail to abstract some of the minutiae:

- **Data Processing:** The `load_data.py` script contains functions to load the `Raw Image Data` and save the image data as numpy arrays into file storage. The `process_data.py` script will load the image data from `data.npy` and preprocess the image by resizing/rescaling the image, and applying filters and ZCA whitening to enhance features. During training the processed image data was split into *training, validation, and testing* data and written to storage. Training also involves a `load_dataset.py` script that loads the relevant data split into a `Dataset` class. For use of the trained model in classifying gestures, an individual image is loaded and processed from the filesystem.
- **Training:** The training loop for the model is contained in `train_model.py`. The model is trained with hyperparameters obtained from a `config` file that lists the *learning rate, batch size, image filtering, and number of epochs*. The configuration used to train the model is saved along with the model architecture for future evaluation and tweaking for improved results. Within the training loop, the training and validation datasets are loaded as `Dataloaders` and the model is trained using *Adam Optimizer* with *Cross Entropy Loss*. The model is evaluated every epoch on the validation set and the model with best validation accuracy is saved to storage for further evaluation and use. Upon finishing training, the training and validation error and loss is saved to the disk, along with a plot of error and loss over training.
- **Classify Gesture:** After a model has been trained, it can be used to classify a new ASL gesture that is available as a file on the filesystem. The user inputs the filepath of the gesture image and the `test_data.py` script will pass the filepath to `process_data.py` to load and preprocess the file the same way as the model has been trained.

3 Sources of Data

3.1 Data Collection

The primary source of data for this project was the compiled dataset of American Sign Language (ASL) called the *ASL Alphabet* from Kaggle user *Akash* [3]. The dataset is comprised of 87,000 images which are 200x200 pixels. There are 29 total classes, each with 3000 images, 26 for the letters A-Z and 3 for *space*, *delete* and *nothing*. This data is solely of the user *Akash* gesturing in ASL, with the images taken from his laptop's webcam. These photos were then cropped, rescaled, and labelled for use.



(a) ASL letter A (b) ASL letter E (c) ASL letter H (d) ASL letter Y

Figure 2: Examples of images from the Kaggle dataset used for training. Note difficulty of distinguishing fingers in the letter E.

A self-generated test set was created in order to to investigate the neural network's ability to generalize. Five different test sets of images were taken with a webcam under different lighting conditions, backgrounds, and use of dominant/non-dominant hand. These images were then cropped and preprocessed.

3.2 Data Pre-processing

The data preprocessing was done using the `PILLOW` library, an image processing library, and `sklearn.decomposition` library, which is useful for its matrix optimization and decomposition functionality.

(a) Sample from non-uniform background test set

(b) Sample from plain white background test set

(c) Sample from a darker test set

(d) Sample from plain white background test set

Figure 3: Examples of the signed letter A T from two test sets with differing lighting and background

Image Enhancement: A combination of brightness, contrast, sharpness, and color enhancement was used on the images. For example, the contrast and brightness were changed such that fingers could be distinguished when the image was very dark.

Edge Enhancement: Edge enhancement is an image filtering techniques that makes edges more defined. This is achieved by the increase of contrast in a local region of the image that is detected as an edge. This has the effect of making the border of the hand and fingers, versus the background, much more clear and distinct. This can potentially help the neural network identify the hand and its boundaries.

Image Whitening: ZCA, or image whitening, is a technique that uses the singular value decomposition of a matrix. This algorithm decorrelates the data, and removes the redundant, or obvious, information out of the data. This allows for the neural network to look for more complex and sophisticated relationships, and to uncover the underlying structure of the patterns it is being trained on. The covariance matrix of the image is set to identity, and the mean to zero.

(a)

(b) Preprocessed Image

Figure 4: Examples of image preprocessing.

4 Machine Learning Model

4.1 Overall Structure

The model used in this classification task is a fairly basic implementation of a Convolutional Neural Network (CNN). As the project requires classification of images, a CNN is the go-to architecture. The basis for our model design came from *Using Deep Convolutional Networks for Gesture Recognition in AmericanSign Language* paper that accomplished a similar ASL Gesture Classification task [4]. This model consisted of convolutional blocks containing two 2D Convolutional Layers with ReLU activation, followed by Max Pooling and Dropout layers. These convolutional blocks are repeated 3 times and followed by Fully Connected layers that eventually classify into the required categories. The kernel sizes are maintained at 3 X 3 throughout the model.

Our originally proposed model is identical to the one from the aforementioned paper, this model is shown in Figure 5. We omitted the dropout layers on the fully connected layers at first to allow for faster training and to establish a baseline without dropout.

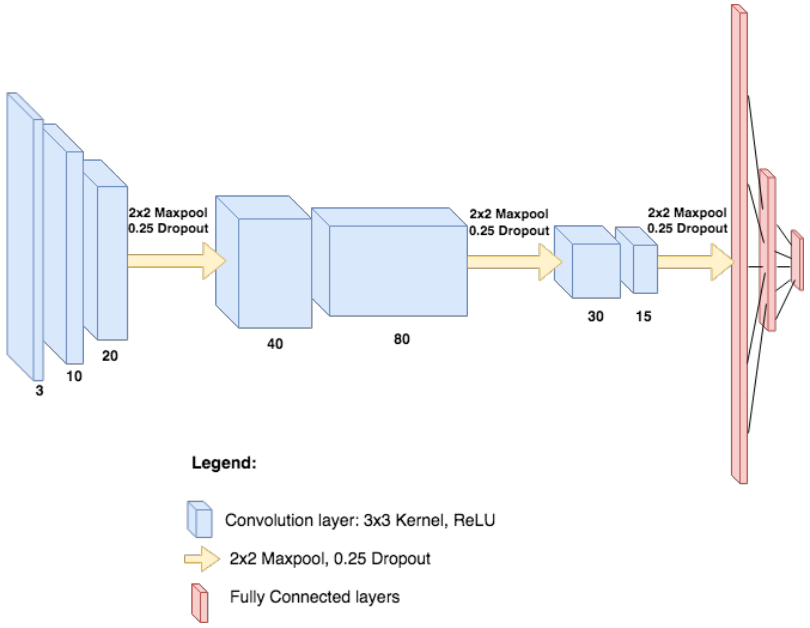


Figure 5: Model Architecture as implemented in *Using Deep Convolutional Networks for Gesture Recognition in AmericanSign Language*[4]

We also decided to design a separate model to compare with the model in the paper. This model was designed to be trained faster and to establish a baseline for problem complexity. This smaller model was built with only one "block" of convolutional layers consisting of two convolutional layers with variable kernel sizes progressing from `5 X 5` to `10 X 10`, ReLU activation, and the usual Max Pooling and Dropout. This fed into three fully connected layers which output into the 29 classes of letters. The variation of the kernel sizes was motivated by our dataset including the background, whereas the paper preprocessed their data to remove the background. The design followed the thinking that the first layer with smaller kernel would capture smaller features such as hand outline, finger edges and shadows. The larger kernel hopefully captures combinations of the smaller features like finger crossing, angles, hand location, etc. This model architecture is shown in Figure 6.

5 Model Performance

5.1 Training and Validation

Our models were trained using *Adam* optimizer and *Cross Entropy Loss*. *Adam* optimizer is known for converging quickly in comparison with Stochastic Gradient Descent (SGD), even while using momentum. However, initially Adam would not decrease our loss thus we abandoned it to use SGD. Debugging Adam optimizer after our final

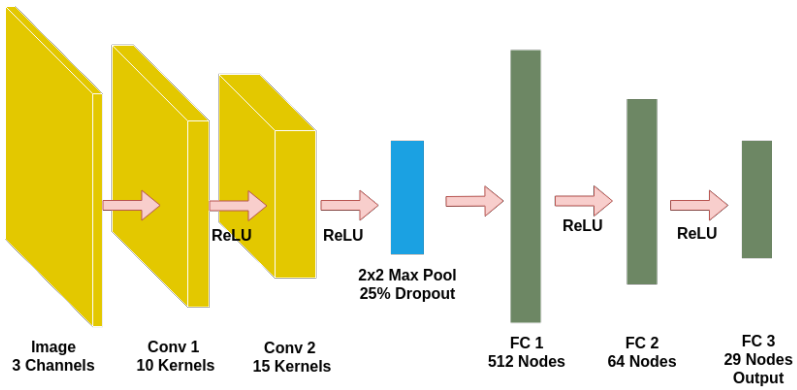


Figure 6: Model Architecture designed for this task for improved training time and establishing a baseline.

presentation taught us that lowering learning rate significantly can help Adam to converge during training. Thus allowing us to train more models towards the end of our project.

Of our two models, the one based on the paper was shown not to be viable, as it took much longer to train without showing any significant decrease in accuracy or loss. We believe this is likely due to the more difficult nature of our classification with the inclusion of background in the images and the lower resolution, causing training to be more difficult. Thus, we decided to focus on improving our smaller model which initially trained to 40% validation accuracy.

Although we had a very large dataset to work with; 3,000 samples for each of 29 classes, after processing the images into numpy arrays, we found our personal computers could load a maximum of 50-100 samples/class and our Google Cloud server could load 200 samples/class. The need to load small datasets actually led us to test the effect of increasing the data available to our models. On our preliminary model, which used strides of 2 on both layers (instead of the strides of 1 and then 2, on our final model) we found the following relation between samples/class and model accuracy:

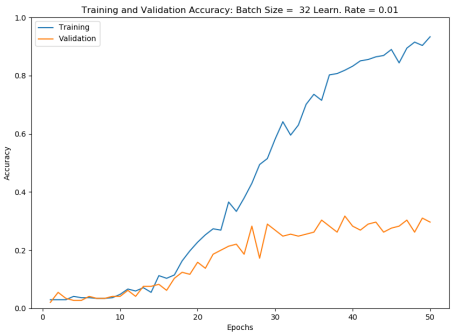
Samples/Class	Avg. Validation Accuracy
25	27.3%
50	34.8%
100	46.1%
200	58.2%

Table 1: Effect of Increasing Dataset Size on Validation Accuracy

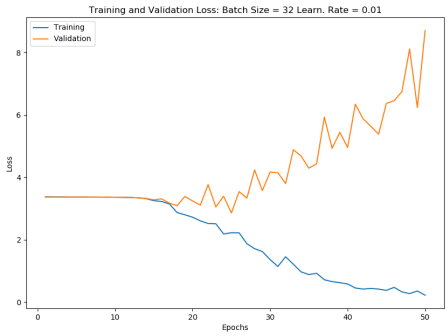
Training our initial models on less data led to the models quickly overfitting as shown in Figure 7. This is likely due to the small amount of samples to train on leading to bad generalization and learning of the sample space.
Increasing the size of our dataset to 200 samples/class led to better model results, with **peak validation accuracy of 60.3%** in epoch 17. However, taking a look at our loss function, we see that the validation loss is increasing, indicating overfitting of the model.
After we applied filtering, enhancement, and ZCA whitening to our dataset, the model performance increased drastically as shown in Figure 9. The **peak validation accuracy achieved is 77.25%** in epoch 24. As shown by the plot of loss, the validation loss is still decreasing, albeit at a slower rate than the training loss, indicating that the model is not drastically overfitting. This shows that preprocessing our images by applying filters and ZCA whitening helps to enhance relevant features for the model to learn.

5.2 Testing

To determine whether our preprocessing of images actually results in a more robust model, we verified on a test set comprised of images from the original dataset, and our own collected image data. The performance of the models on the test set is shown in Table 2. We see that the model trained on preprocessed images performs much better than the model trained on the original images, likely due to the former's lack of overfitting.
Taking a look at the confusion matrix for the Filtered Model on the Kaggle test set in Figure 10, we can see that

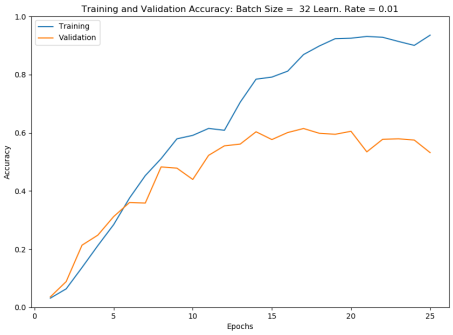


(a) Training and Validation Accuracy

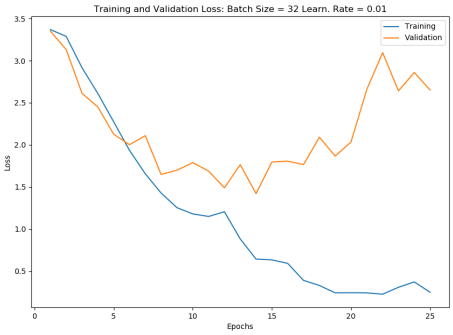


(b) Training and Validation Loss

Figure 7: Training and Validation Performance of Initial Model Trained on Small Dataset of 50 Samples/Class Demonstrating Overfitting



(a) Training and Validation Accuracy



(b) Training and Validation Loss

Figure 8: Training and Validation Performance of Model Trained on Original Images

the model is fairly good at predicting the correct letter. Taking a look at some of confused letters, like V and W in Figure 11, we see that the two letters are similar in shape. This provides us with some intuition about why the model might confuse these two letters, as a W looks like a repeated V.

6 Ethical Issues

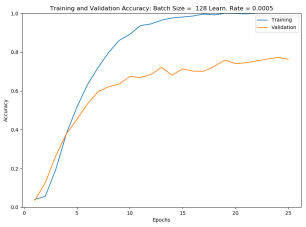
A few ethical issues:

- This technology is one that allows a disabled community to further integrate into an abled community, and may be viewed as an assimilation and bending to the rules of privileged community. This may reduce efforts of hearing people to accommodate for deaf people.
- The dataset needs to be diverse enough in order to accommodate people of all skin tones and in all environments. A bias in data could possibly disadvantage deaf people of a certain ethnic group.

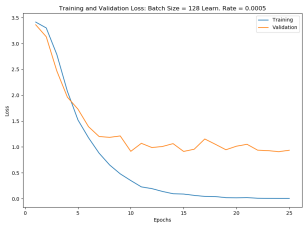
7 Reflection

7.1 Key Learnings

- Train Early and Often: Due to the long training time of our model, it became cumbersome to test various hyperparameters, architectures, image filtering, etc. In future projects, training earlier can identify such issues and more time can be dedicated to training.



(a) Training and Validation Accuracy



(b) Training and Validation Loss

Figure 9: Training and Validation Performance of Model Trained on Filtered Images

Category	Unfiltered Model	Filtered Model
Kaggle Test Accuracy	76.8%	62.1%
Collected Test Accuracy	27.1%	8.03%

Table 2: Effect of Increasing Dataset Size on Validation Accuracy

- Experiment With Optimizers: Due to our initial difficulties with *Adam*, we abandoned it for the slower *SGD*. However, further testing with *Adam* and varying our hyperparameters allowed Adam to converge. The lesson to take away is to try various optimizers early on to establish which ones converge the quickest for faster training.
- Amount of Data: As shown in Section 5.1, increasing the training set size leads to drastic improvement in model performance. This likely increases the robustness of the model through learning more of the possible sample space.

7.2 Future Steps

- Use Dynamic Loading for Dataset: Our original dataset was quite large and is impossible to use without a server with a lot of RAM and disk space. A possible solution is to split the file *names* into training, validation, and test sets and dynamically loading images in the `Dataset` class. Using such a loading technique would allow us to train the model on more samples in the dataset.

References

[1] Farnaz D. Notash and Elahe Elhamki. "Comparing loneliness, depression and stress in students with hearing-impaired and normal students studying in secondary schools of Tabriz". In: *International Journal of Humanities and Cultural Studies* February 2016 Special Issue (2016). ISSN: 2356-5926.

[2] "The Cognitive, Psychological and Cultural Impact of Communication Barrier on Deaf Adults". In: *Journal of Communication Disorders, Deaf Studies Hearing Aids* 4 (2 2016). DOI: 10.4172/2375-4427.1000164.

[3] Akash. *ASL Alphabet*. URL: https://www.kaggle.com/grassknoted/asl-alphabet. (accessed: 24.10.2018).

[4] Vivek Bheda and Dianna Radpour. "Using Deep Convolutional Networks for Gesture Recognition in American Sign Language". In: *CoRR* abs/1710.06836 (2017). arXiv: 1710.06836. URL: http://arxiv.org/abs/1710.06836.

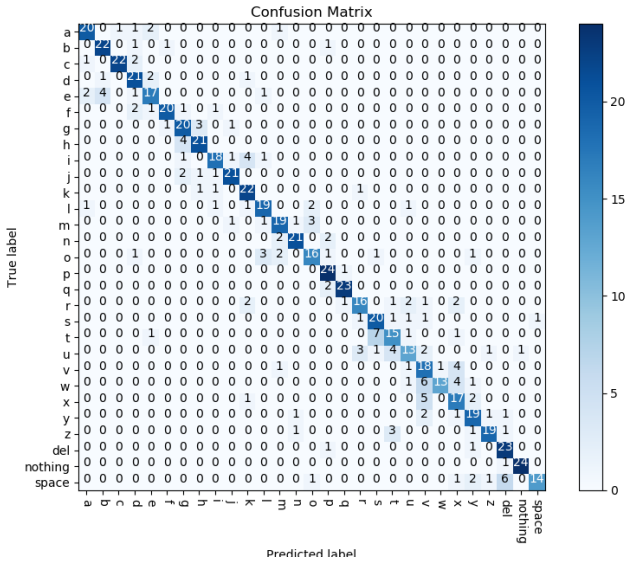


Figure 10: Confusion Matrix for Filtered Model on Kaggle Test Set

(a) Letter V

(b) Letter W

Figure 11: Showcasing similarity between the ASL gesture for V and W