ASL Sign language Prediction

Pratik Jogdand[1] **Atharva Muley**[2]

Abstract

Speech and hearing impaired people use sign language as an alternative to traditional vocal for of communication to communicate. In the United States of America, ASL is widely used by these specially challenged people. But, not many of us know how to communicate in ASL language. To solve this problem, we have tried to create a machine learning model which acts as a bridge of communication between hearing impaired people and normal people. The model will convert ASL into ASCII so that normal people will be able to communicate with ASL users without having to learn the ASL language itself.

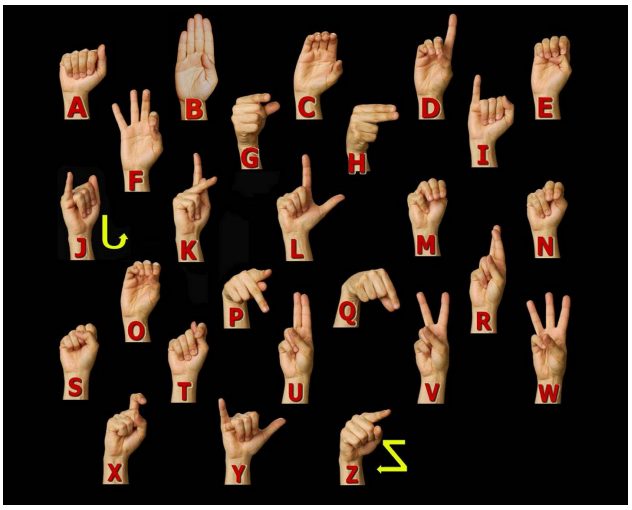

Figure 1. ASCII values and ASL counterparts (bab)

1. Introduction

ASL (American Sign Language) is a Sign Language used by deaf communities to communicate primarily in the United States of America and some parts of Canada. With some slight variation of the ASL, it can also be found in other continents including West Africa and Southeast Asia. Other countries also have their own variations of the sign language such as the British Sign Language (BSL), Japanese Sign Language and the Spanish Sign Language et cetera (Kar & Chatterjee, 2015).

The language is made up of different variations of finger positioning humans can do and every position is unique and easy to perform at the same time. Every position is related with a respective alphabet to prevent any confusion. Apart from this, it also follows its own Grammar rules. Unfortunately right now the language is understood by a small percentage of all the people apart from the deaf community. Thus, giving rise to the gap for effective communication between the deaf community and the people who don't know ASL language.

[1]University of Maryland, Baltimore County [2]. Correspondence to: Pratik Jogdand <pratikj1@umbc.edu> .

University of Maryland, Baltimore County. Correspondence to:Atharva Muley<amuley1@umbc.edu>.

We were motivated by the fact that AI can be used to solve this problem and diminish the present language gap. Our approach is quite simple and it was to Build a machine learning model which is capable of inferring signs and then generating text, which will eventually make this gap smaller. This can be useful in commercial places like restaurants where the deaf people can place an order without requiring the cashier to know ASL Sign language or vice versa.

1.1. Objective

To train a deep learning model for predicting ASCII values of ASL Sign language with additional space, delete and nothing.

2. Related Work

Many people have studied about ASL and how we can infer ASL using modern technologies. As in any system, there are multiple steps to convert ASL into normal language if we are using any technology. Aradhana Kar et al came up with an efficient way to convert ASL into normal language using Video processing (Kar & Chatterjee, 2015). They were able to convert ASL into simple English sentences in an optimised and quicker way. The closest study we found to our similar approach was by Prateek SG et al who

used Convolution Neural Networks to create a Dynamic tool interpreter and achieved 98.66% Accuracy (S.g., 2018). Another approach to interpret ASL dynamically was done using leap motion controller was done by Deepali Naglot and Miling Kulkarni (Naglot & Kulkarni, 2016). We tried to identify ASL alphabets whereas Md. Shahin Alom et al tried to identify ASL digits using Convolution Neural Networks and Support Vector Machines (Alom, 2019).

3. Problem Definition

3.1. Dataset

The first challenge we came across was getting a dataset which had ASL in the form of pictures. Fortunately we found a dataset on kaggle which aligned with our requirements.

The training data set contains 87,000 images. There are 29 classes, of which 26 are for the english alphabets and 3 classes for space, delete and nothing respectively. The test dataset has 29 images each for one class. The train images are organized in their individual folders denoting the image's class. There are a total for 87000 training images. The images are 200 x 200 pixels wide RBG images. The images are taken in various lighting conditions to add variation in data.

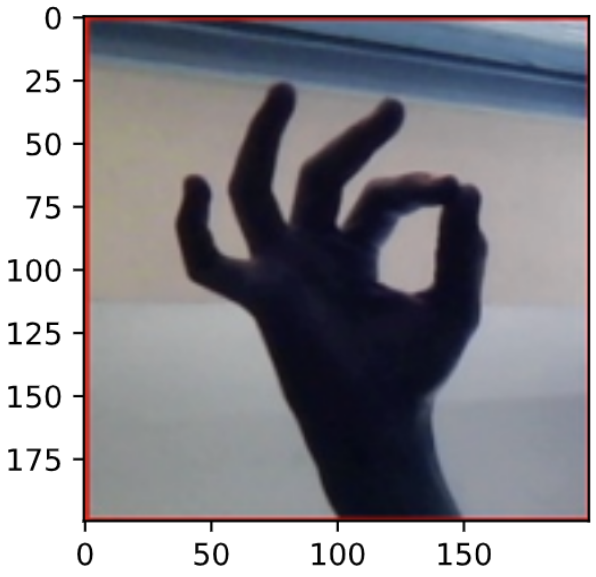

Figure 2. Sample Image from dataset

3.2. Model

We built our model in Keras and used Tensorflow as a backend. We build a Sequential model and added bunch of different layers to it[4]. The input shape for the model was set to 128, 128. All the convolution layers has 4 Kernels. And we used ReLU as the activation function in all the layers. We stacked multiple Convolution layers with increasing number of filters as we go deeper in the model. Adding more filters helped to represent the intermediate sub-features of the original image with more details. Then we used Flatten to convert the last Conv2d to one dimensional vector and feed it to fully connected layer. We then defined two fully connected layers with 128 and 64 neurons. The output layer had 29 neurons representing each class and softmax function to normalize the probabilities of all classes so that they add up to 1.

4. Proposed Method

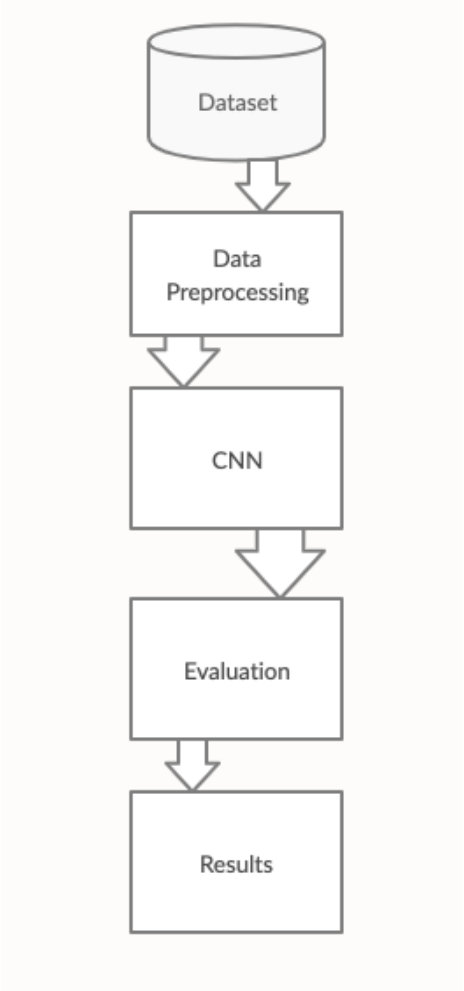

Figure 3. Final Model Summary

The diagram represents out approach to create the CNN and then to evaluate it. In the preliminary stages we fetched the data-set and prepared it so that it is ready for training. After designing and developing the CNN, it was tested against train, validation and test data sets. The observations were

evaluated and finally the project is concluded.

4.1. Preparing the Dataset

In the root directory of the images, they are already organized in their individual folders respectively. To read the images, we used the Keras ImageDataGenerator class as it is one of the best if not the best way to read the images for further computational work. Using Keras, the images were re-scaled to 128 x 128 x 3 pixels before normalisation. And then they were normalized.
The next step was to split the data into training and validation sets so what we will be able to experiment with training set to train our model. For this, all of the 87,000 images were then split into training and validation set with 78,300 and 8,700 images respectively. Now coming to the test set, it only has one image from every class we tried to classify i.e 29 images.

To train the model efficiently and fairly, the batch generator generated a batch of 64 images which were randomly picked from the training set.

4.2. Training

We used Google Colab to train our model and chose python as the language of our preference for coding the model. We used Keras fit method to train the model using ADAM optimizer with a learning rate of 0.001 and categorical cross-entropy as model's loss. With a batch size of 64 and 78,300 training images the model took 1224 steps per epoch each iteration. Our final model was iterated over entire dataset 15 times.

5. Experiments

We did many train/dev iterations with different model configurations both varying one hyperparameter while keeping others constant and varying multiples at a time. With various configurations and their respective results to chose from we selected the model that gave the best accuracy over the validation set as it has the maximum chances to function optimistically with any test set.

For brevity we have only listed few of our train/dev iterations in the table below:

Model #	Train Accuracy	Validation Accuracy
1	98.3%	79.73%
2	96.98%	87.83%
3	92.75%	90.56%
4	95.96%	93.17%

Table 1. Model Comparisons

The image below describes the final model summary which was used to conduct the experiments mentioned in this section.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 125, 125, 64)	3136
dropout_1 (Dropout)	(None, 125, 125, 64)	0
conv2d_2 (Conv2D)	(None, 61, 61, 64)	65600
dropout_2 (Dropout)	(None, 61, 61, 64)	0
conv2d_3 (Conv2D)	(None, 58, 58, 128)	131200
dropout_3 (Dropout)	(None, 58, 58, 128)	0
conv2d_4 (Conv2D)	(None, 28, 28, 128)	262272
dropout_4 (Dropout)	(None, 28, 28, 128)	0
conv2d_5 (Conv2D)	(None, 25, 25, 128)	262272
dropout_5 (Dropout)	(None, 25, 25, 128)	0
flatten_1 (Flatten)	(None, 80000)	0
dropout_6 (Dropout)	(None, 80000)	0
dense_1 (Dense)	(None, 128)	10240128
dropout_7 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 29)	1885

Total params: 10,974,749
Trainable params: 10,974,749
Non-trainable params: 0

Figure 4. Final Model Summary

In the initial stages of the project, our first model was overfit. The
but on the other hand it could only achieve approx 80% on dev set. To fix this we added multiple dropout layers to the model and saw an improvement of 8% to the validation accuracy. We then added L2 Regularization to the 2 fully connected layers and saw another increase int the validation accuracy by approximately 3%. Finally, for training our final model we increased the epochs from 5 to 15 and saw another 3% increase in the validation accuracy. After adding regularization the difference between the training and validation test was reduced and the model was not overfit.

6. Observations

As mentioned earlier, cloud platform google colab was used to design and develop this model and thus all the observations in this section were a result of using a Nvidia V100 GPU made available by google on its cloud platform.

6.1. Evaluation Metrics

To evaluate our machine learning models, we have to find the accuracy across the training, validation and test sets.

- Accuracy - It can be calculated with by summing the true positive values with true negetive values and then divide it with the sum of total number of postives and negetives present in the data set. (Ahmad, 2018)

$$Accuracy = \frac{TP + TN}{P + N}$$

Accuracy of the model can be figured out using the confusion matrix made below. All the mentioned terms in the confusion matrix can explained as follows (def).

- True Positive (TP) - These are the values which were correctly predicted by our machine learning models.
- False Positive (FP) - These are the values which were negative according to the data set, but were predicted as positive by our machine learning models.
- False Negative (FN) - These values are predicted values by our models as negatives but they are true according to the data set.
- True Negative (TN) - They are the correctly predicted negative values by our machine learning models.

		Prediction outcome		
		p	**n**	**total**
actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

Figure 5. The confusion Matrix

6.2. Observations on Train and Validation sets.

In the initial stages of the project, our model had only single fully connected layer. But having only one layer resulted in some over-fitting. Adding another fully connected layer helped model to fit the complexities of the data and we observed a huge jump in accuracy.

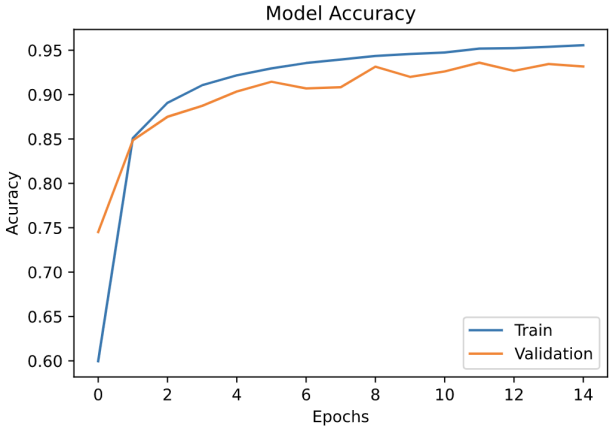

Figure 6. Final Model Accuracy

As you can see from the above image, the accuracy goes up exponentially in the second epoch itself following with a gradual ascent in accuracy in the coming epochs. After many trials and tests, we were able to achieve maximum accuracy before it started to dip after some epochs and the model was able to converge. Here we can observe that the model is learning well and there is not much difference between the accuracy of the two.

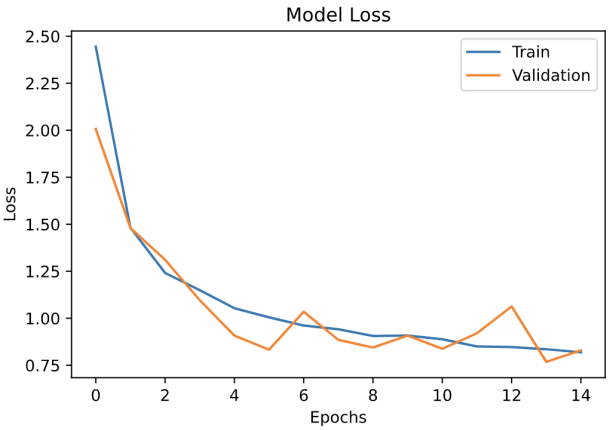

Figure 7. Final Model Loss

With the increase in accuracy, the loss also decreases as expected. The validation loss does wobbles further in the training but controlled it by early stopping and set this as our convergence criteria. Thus, we were able to achieve the results efficiently in less number of epoch as projected earlier.

For certain model configurations, the Validation loss was wobbling hinting that the model overshot the maxima.

Figure 5 below depicts the change in accuracy over train and validations sets and how it is sensitive to number of epochs.

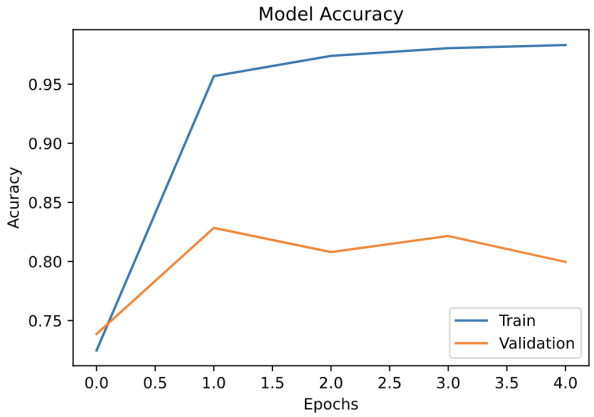

Figure 8. Baseline Model Accuracy

Figure 6 shows how the model's loss reacted to different number of epochs until convergence.

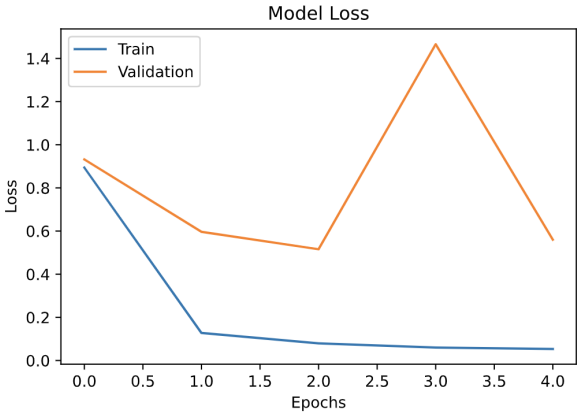

Figure 9. Baseline Model Loss

6.3. Evaluating the Test set

As mentioned earlier, the dataset has a single test image for each class. Our model predicted 26 images correctly out of 29 images thus, the accuracy of test set was 89.65%. The only three images our model was not able to predict in this specific attempt on test set were T, M and N as the model confused it with S and A respectively.

7. Conclusion

We implemented a deep learning model that is capable of classifying the images that are fed to it. After many trails and errors, we were able to achieve a training accuracy of 95.98%, validation accuracy of 93.17% and test accuracy of 89.65% respectively. We tried multiple different model configurations to find out the best of all. We also saw how adding layers and their tuning their parameters and regularization plays an important role in reducing the over-fitting and increasing the models overall accuracy.

Acknowledgements

We would like to thank our Professor Dr. Tim Oates to present us with this opportunity to create and deploy a Convolution Neural Network. We have learned both practical usage of ML and theoretical aspects such as writing a paper by completing this project.

References

http://lifeprint.com/as1101/topics/wallpaperl.html.

https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/.

Ahmad, Iftikhar, e. a. Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. In *IEEE Access, vol. 6, 2018, pp. 33789–33795., doi:10.1109/access.2018.2841987*, 2018.

Alom, Md. Shahin, e. a. Digit recognition in sign language based on convolutional neural network and support vector machine. In *International Conference on Sustainable Technologies for Industry 4.0 (STI) doi:10.1109/sti47673.2019.9067999*, 2019.

Kar, A. and Chatterjee, P. S. An approach for minimizing the time taken by video processing for translating sign language to simple sentence in english. In Langley, P. (ed.), *International Conference on Computational Intelligence and Networks, Bhubaneshwar*, pp. pp. 172–177, 2015.

Naglot, D. and Kulkarni, M. Real time sign language recognition using the leap motion controller. In *International Conference on Inventive Computation Technologies (ICICT) doi:10.1109/inventive.2016.7830097*, 2016.

S.g., P. Dynamic tool for american sign language finger spelling interpreter. In *International Conference on Advances in Computing, Communication Control and Networking (ICACCCN) doi:10.1109/icacccn.2018.8748859*, 2018.