Interpreting Sign Language using Image Classification Techniques

Mustafa Khan
Department of Computer Science
Stanford University
mkhan7@stanford.edu

Amita Gondi
Department of Computer Science
Stanford University
agondi@stanford.edu

I. Abstract

A real-time sign language translator is an important milestone in facilitating communication between the deaf community and the general public. We hereby present the development and implementation of an American Sign Language (ASL) fingerspelling translator based on a convolutional neural network. We utilize the American Sign Language MNIST dataset to produce a robust CNN model that consistently classifies hand-gesticulated letters correctly in a majority of cases (with the limitation of J and Z, which require hand motion and thus are not easily handled by an image classification model). Given the limitations of the datasets and the encouraging results achieved, we are confident that with further research and more data, we can produce a fully generalizable translator for all ASL letters.

II. Introduction

Approximately 500,000 documented people use American Sign Language (ASL) to communicate, and up to one million more use it as their primary or secondary language. Beyond this group, people born with hearing or speaking disabilities find it difficult to communicate with the rest of the population. The primary factor here is education, with ASL not a widely standardized language and inaccessibility prevents it from being picked up at later stages. This creates a mismatch in communication that is difficult to bridge.

We look to devise a baseline using logistic regression and thereby, use a convolutional neural network (CNN) based model via which we can classify images of hand-gestures signing the various letters in the language correctly, with the goal one day of a generalizable translator that may be easily used with one's smartphone and camera.

III. Related Work

Since the 2000s, researchers have used classifiers from a variety of categories that we can group roughly into linear classifiers, neural networks and Bayesian networks for ASL recognition [1-11]. While linear classifiers are easy to work with because they are relatively simpler models, they require sophisticated feature extraction and preprocessing methods to be successful. Singha and Das [1] obtained accuracy of 96 percent on 10 classes for images of gestures of one hand using Karhunen-Loeve Transforms where they use a skin filter, hand cropping and edge detection and then translate and rotate the axes to establish a new coordinate system based on the variance of the data. Their model simply distinguishes between hand gestures like thumbs up and numbers (no ASL). Research like Sharma et al. [12] categorize a color channel using Support Vector Machines and k-Nearest Neighbors (grouped under piece-wise classifiers). Achieving a 62.3 percent accuracy on color segmentation, the model uses a contour trace. However, ASL requires the need to capture continuous video stream as well - particularly for letters j and z. Suk et al. propose a method using a dynamic Bayesian network [13] where they classify moving hand gestures and achieved an accuracy of over 99 percent, but on gestures markedly different from each other and not in ASL. Moving toward neural networks, we see that they are best at learning features for classification but take significantly more time and data to train. Mekala et al. classified video of ASL letters into text using advanced feature extraction and a 3-layer Neural Network [3] where they extracted features in two categories: hand position and movement. Prior to ASL classification, they identify the presence and location of 6 points of interest in the hand: each of the fingertips and the center of the palm. Mekala et al. also take Fourier Transforms of the images and identify what section of the frame the hand is located in. While they claim to be able to correctly classify 100 percent of images with this framework, there is no mention of whether this result was achieved in the training, validation or test set. The most relevant work to date is L. Pigou et als application of CNNs to classify 20 Italian gestures using Microsoft Kinect (captures depth and aids in classification) on people performing full-body gestures and achieves a cross-validation accuracy of 91.7 percent.

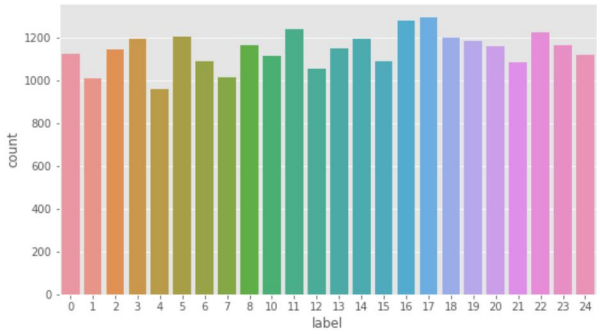
IV. Dataset and Features

We used the Sign Language MNIST dataset available at kaggle.com/datamunge/sign-language-mnist in our training and experiments. It contains $27,455$ training samples and $7,172$ test samples. Each of these samples consists of a grayscale 28x28 pixel image with values ranging from $0-255$, as well as a corresponding numerical label for that image from $0-25$, each matching a letter from the English alphabet (0 corresponds to A, 1 to B, and so on). Note, however, that

we do not expect strong results for J (9) and Z (25) in our experiments as these two letters require motion of the hands to sign, and will thus be excluded as they cannot be properly detected from a still image.

Below is a grayscale image montage panel of each letter labeled and 28x28 pixels with (0-255) pixel values.

And here is the full breakdown of the label counts over our entire dataset.



The images for the dataset consist of multiple pictures taken for each of the various classes with variation in terms of background, lighting conditions, and the person signing the alphabets. The dataset is further augmented using various data augmentation techniques to change the lighting, crop the image, and resize to the effect of introducing generalizability to the model.

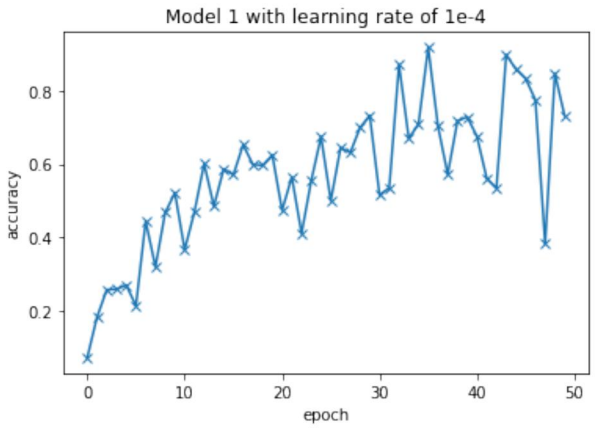
V. Methods Considered

A. Baseline: Logistic Regression

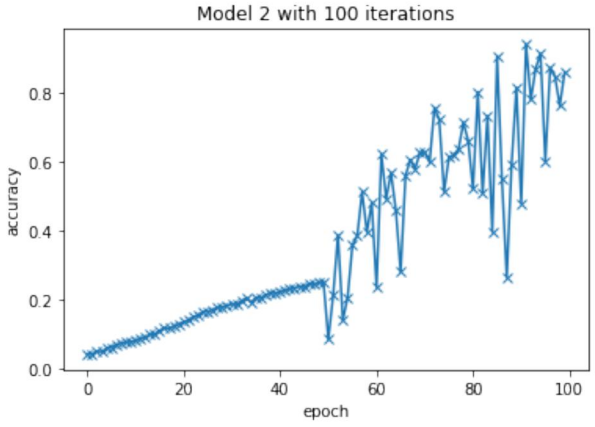
To construct our model, we used PyTorch where the pixel values are multiplied by a weight and added by some offset bias that affects our letter classification. Given that we have 26 outputs, instead of the typical 0 or 1, PyTorchs linear function will output a tensor with 26 elements with each element denoting the probability that the image is that symbol. This is done after normalizing the array using PyTorchs $cross_entropy$ function. For example, if the output of the function is [0.0001, 0.003, ..., 0.007], then there is a 0.0003 chance that the letter in the hand image is B.

1) Training the model: To start out training the model, we will consider one learning rate of 1e-4 and train two separate models, one for 50 and the other for 100 iterations. Our points of interest are the accuracy and validation losses (in that order).

For Model 1, we train on 50 iterations, and start out with a low accuracy as we chose random weights and biases. We printed out the validation loss and accuracy on every iteration and then plotted it. We see that with this learning rate, it gradually increases although there are some big blips with the iterations and we end up with a training accuracy of 0.85. It is worth noting that this model has yet to plateau, so we increased our training duration.

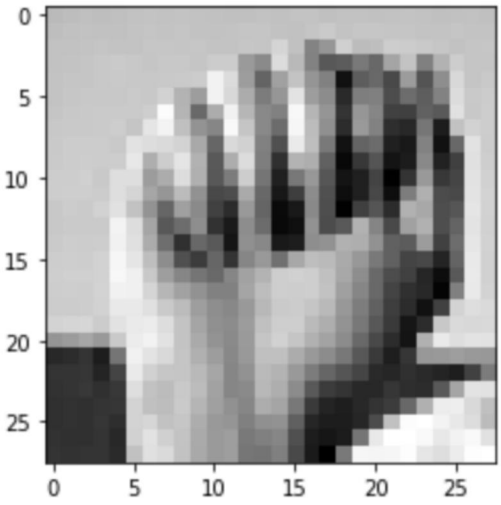


In Model 2, with an increase to 100 iterations to train on, we see that with the same learning rate, it increases slower than Model 1 before becoming very volatile but also plateaus with a training set accuracy of 0.88.



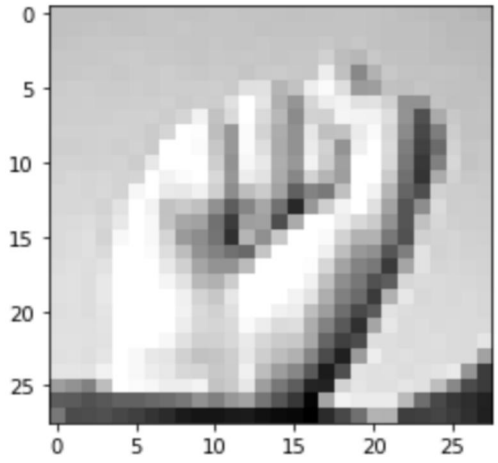
2) Model Prediction: With Model 1, we see that the accuracy on the test dataset was 0.62, and a validation loss of 17.14. And with model 2, we see a very similar trend with the accuracy on the test dataset was 0.62, and a validation loss of 17.20.

```
Label: A , Predicted: A
```



```
Label: S , Predicted: V

<matplotlib.image.AxesImage at 0x7f
```



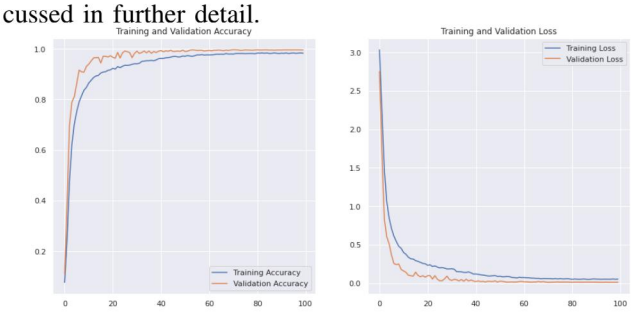
Some error analysis yields the observation that the model could possibly optimize for the concentration of dark pixels in the middle, which can be confused for multiple alphabets (and especially in a low pixelated image such as that provided in our dataset). It might be worth looking into other image datasets available online as well as being more careful about data augmentation techniques.

B. Convolutional Neural Network

To improve on the baseline, we wanted to see what the potential for improvement was with a convolutional neural network model. CNNs are known to be adept at image detection and classification. The input to the model is taken to be a tensor with shape (num inputs) x (input height) x (input width) x (input channels), with each image abstracted into the activation map by passing through the convolutional layer. This has the shape of (num inputs) x (feature map height) x (feature map width) x (feature map channels). The label dataframe consists of single values from 1 to 26 for each individual picture. Accordingly, the output layer in the CNN consists of 26 nodes, and each integer is encoded in a one-hot array of size 26.

1) Training the model: The model itself is Sequential, with layers consisting of a convolutional layer with input shape $(28, 2, 8, 1)$ and ReLU activation function, followed by a MaxPool and Dropout layer both (to reduce the dimensionality of the data) with dropout probability 0.2. All of this is repeated thrice. This is then followed by a Flatten layer and then four Dense layers (the first three with activation ReLU and of sizes 4096, 1024, and 256). The fourth is of size 25 with activation softmax). The model is then trained using the Adam Optimizer using sparse categorical cross entropy loss, optimizing for accuracy with a batch size of 128. We then trained the model over 100 epochs.

We further tuned the hyperparameters, which is later discussed in further detail.



2) Model Prediction: We achieved very good results with a training accuracy of 0.996, validation accuracy of 0.958, and test accuracy of 0.989.

We built a confusion matrix with the precision, recall, and f-1 score of each class to further understand our model, which is attached below. It is visible that our model lags in results on 6 in particular, as well as on 13, which correspond to the letters G and O, respectively. Though we still get good results on both of these, it appears that our model casts a smaller net for G, achieving high precision, but also missing some cases. The reverse is true for O for which the model (though continuing to perform well) has a 0.93 precision but perfect recall, indicating it inadvertently catches a few false positives for this class.

C. Hyperparameter Search to maximise validation accuracy

The choice of model and choice of hyperparameters are perhaps the most important part of using the above techniques effectively. Hence, in order to make an informed choice, we need a way to validate that our model and our hyperparameters are a good fit to the data.

As we are not interested in looking at how just one parameter changes, but how multiple parameter changes can affect our results, we do cross-validation with more than one parameters simultaneously, effectively trying out combinations of them. Here, we multiply the number of features to validate to see how many combinations there are where each combination is evaluated using the k-fold cross-validation (k is a parameter we choose).

	precision	recall	f1-score	support
0	0.99	0.99	0.99	331
1	1.00	1.00	1.00	432
2	1.00	0.94	0.97	310
3	1.00	1.00	1.00	245
4	0.99	1.00	1.00	498
5	1.00	1.00	1.00	247
6	1.00	0.93	0.97	348
7	1.00	1.00	1.00	436
8	1.00	1.00	1.00	288
9	1.00	0.98	0.99	331
10	1.00	1.00	1.00	209
11	1.00	0.99	1.00	394
12	1.00	1.00	1.00	291
13	0.93	1.00	0.96	246
14	0.95	1.00	0.98	347
15	1.00	1.00	1.00	164
16	0.98	1.00	0.99	144
17	1.00	1.00	1.00	246
18	0.95	1.00	0.97	248
19	1.00	0.98	0.99	266
20	1.00	1.00	1.00	346
21	1.00	1.00	1.00	206
22	1.00	1.00	1.00	267
23	1.00	1.00	1.00	332
accuracy			0.99	7172
macro avg	0.99	0.99	0.99	7172
weighted avg	0.99	0.99	0.99	7172

For our CNN, we perform a GridSearch for batch size, number of epochs and initializer combined to search for different values where the choices are specified into a dictionary and passed to GridSearchCV which performs an exhaustive search over the parameters specified to ”fit” and ”score” the best combination via cross validation.

We experimented with batch sizes of 128, 256, and 512, and epoch sizes of 20, 50, and 100 to identify the optimum training method. Ultimately, we achieved the best results with a smaller batch size of 128 and training for 100 epochs by which point our training and loss curves levelled out. We also identified that a random probability of 0.2 in the dropout layer was sufficient to increase the generalizability of the overall learned function.

VI. Results and discussion

For our baseline, we saw that Logistic regression can give us a test accuracy hovering around 62 percent. From the performance of our models, increasing the number of training iterations slightly increases training accuracy but plateaus at 85 percent. We look to continue tweaking the baseline as well as potentially considering SVM to serve as a secondary baseline.

In terms of the primary model, we are satisfied by the results of the CNN which has been optimized fully to the needs of the dataset. At the same time, we recognize limitations such as the straightforward nature of the images in the dataset, rather than presented from different angles or potentially blurred. We also recognize the low resolution of the images in the dataset, and further work would entail generalizing the model to different datasets.

As mentioned earlier, the exclusion of J and Z from specific analysis is worth considering in light of the idea that these letters are dependant on hand motion. Given time, it would be worth exploring k-NN to classify these images so as to provide a holistic analysis on different machine techniques in the context of ASL hand sign classification.

Lastly, we explored model validation and hyperparameter optimization, focusing on cross-validation with more than one hyperparameters and how it comes into play when fitting models to data. In particular, we found that a smaller batch size performed best in this regard and provided the optimal results, along with a smaller dropout size of 0.2.

We do note some future work that could be done for a production-ready pipeline - we could combine overlapped detection patches and not have 30 detections of the same letter, but reduce overlapping groups of detections down to a single one. This could be done via an unsupervised clustering approach (MeanShift Clustering is one good candidate for this), or via a procedural approach such as non-maximum suppression, an algorithm common in machine vision.

VII. Conclusion

The relevant work for this paper can be accessed on Github at github.com/mustafakhan14/SignLanguage_Classifier. Note that it is essential to access the data from the Kaggle dataset at kaggle.com/datamunge/sign-language-mnist. The results of these experiments are exciting and show potential for implementation to facilitate day to day communication. Beyond American Sign Language, future work can also look into training models for other sign language translation systems such as the British, Arabic, Japanese, and Chinese versions.

VIII. Contributions

Mustafa Khan Built the convolutional neural network architecture. Worked on data collection and augmentation in addition to appropriate experiments.

Amita Gondi Built algorithm for hyperparameter search, implemented logistic regression baseline classifier and conducted experiments to both optimize the model and understand its results.

References

[1] Singha, J. and Das, K. Hand Gesture Recognition Based on Karhunen-Loeve Transform

[2] D. Aryanie, Y. Heryadi. American Sign Language-Based Finger-spelling Recognition using k-Nearest Neighbors Classifier

[3] P. Mekala et al. Real-time Sign Language Recognition based on Neural Network Architecture.

[4] J. Atwood, M. Eicholtz, and J. Farrell. American Sign Language Recognition System.

[5] B. Garcia and S. Viesca, “Real-time American Sign Language Recognition with Convolutional Neural Networks’

[6] L. Pigou et al. Sign Language Recognition Using Convolutional Neural Networks.

[7] Lifeprint.com. American Sign Language (ASL) Manual Alphabet (fingerspelling) 2007.

[8] G. Saggio, "Sign Language Recognition Using Wearable Electronics: Implementing k-Nearest Neighbors with Dynamic Time Warping and Convolutional Neural Network Algorithms"

[9] F. Utaminingrum, "Alphabet Sign Language Recognition Using K-Nearest Neighbor Optimization "

[10] R. Dias , L. Fontenot , K. Grant , C. Henson , and S. Sood ., "American Sign Language Hand Gesture Recognition" unpublished.

[11] R. Sharma et al. Recognition of Single Handed Sign Language Gestures using Contour Tracing descriptor.

[12] H. Suk et al. Hand gesture recognition based on dynamic Bayesian network framework. Patter Recognition 43 (9); 3059-3072, 2010.