Real-time American Sign Language Gesture Recognition using Convolutional Neural Networks (CNNs)

Ronil Christian
Illinois Institute of Technology
Chicago, IL, USA
rchristian@hawk.iit.edu

Abstract

American Sign Language has become a complex and important means for communication for the deaf and dumb community. With the increasing popularity of technology, the need has arisen for the development of automatic recognition of ASL gestures. Hereby presented is a deep learning approach for the classification of ASL gestures using convolution neural networks (CNNs). Overall, this paper aims to show promise for automatic recognition of ASL gestures which is fully generalizable over a wide range of applications.

Keywords: *Convolutional Neural Networks, Deep Learning*

1. Introduction

American Sign Language (ASL) is a visual language that is used by the deaf and hard of hearing community in the United States and Canada. According to the National Institute on Deafness and Other Communication Disorders, approximately 15% of American adults (37.5 million people) report some trouble hearing, and around 2 to 3 out of every 1,000 children in the United States are born with a detectable level of hearing loss.

ASL gesture classification remains a challenging problem due to the complexity of ASL gestures and the high degree of variation in signing styles among individuals. Automatic recognition of ASL gestures has the potential to provide a more efficient and accurate means of communication for the deaf and hard of hearing community.

This paper intends to present an ASL recognition system that uses Convolutional Neural Networks (CNN) in real time to translate a video of a user's ASL signs into text. Our problem consists of three tasks to be done in real time:

1. Obtaining video of the user signing (input)
2. Classifying each frame in the video to a letter
3. Reconstructing and displaying the most likely word from classification scores (output)

From a computer vision perspective, this problem represents a significant challenge due to a number of considerations, including:

- Environmental concerns (e.g., lighting sensitivity, background, and camera position)
- Occlusion (e.g., some or all fingers, or an entire hand can be out of the field of view)
- Sign boundary detection (when a sign ends and the next begins)
- Co-articulation (when a sign is affected by the preceding or succeeding sign)

While Neural Networks have been applied to ASL letter recognition in the past with accuracies that are consistently over 90%, many of them require a 3-D capture element with motion-tracking gloves or a Microsoft Kinect, and only one of them provides real-time classifications. The constraints imposed by the extra requirements reduce the scalability and feasibility of these solutions.

Our system features a pipeline that takes video of a user signing a word as input through a web application. We then extract individual frames of the video and generate letter probabilities for each using a CNN (letters a through y, excluding j and z since they require movement). With the use of a variety of heuristics, we group the frames based on the character index that each frame is suspected to correspond to. Finally, we use a language model in order to output a likely word to the user.

2. Related Work

Researchers have used a variety of classifiers like liner classifiers, neural networks, and Bayesian networks.

Recent studies have shown that deep learning approaches, particularly CNNs, can achieve state-of-the-art performance in gesture recognition tasks. For example, in a study by Donahue et al. (2015), a deep learning approach was used to classify ASL gestures with an accuracy of 90.1%. Similarly, in a study by Zhang et al. (2017), a CNN-based approach achieved an accuracy of

96.7% on a dataset of Chinese Sign Language (CSL) gestures.

Other studies have focused on improving the robustness of gesture recognition systems by incorporating multiple modalities, such as depth information and skeletal tracking. For example, in a study by Hu et al. (2018), a multimodal approach was proposed that combined RGB images and depth maps for recognition of ASL gestures.

Mekala et al. classified video of ASL letters into text using advanced feature extraction and a 3-layer Neural Network. 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.

3. Approach and Methods

3.1. Classifier Development

Since Convolutional Neural Networks (CNNs) have seen incredible success in handling tasks related to processing images and videos, this paper adopts the same machine learning algorithm to classify the ASL gestures.

A primary advantage of utilizing such techniques stems from CNNs abilities to learn features as well as the weights corresponding to each feature. Like other machine learning algorithms, CNNs seek to optimize some objective function, specifically the loss function.

Using a softmax-based classification head allows to output values akin to probabilities for each ASL letter. These probabilities afforded to us by the softmax loss allows to more intuitively interpret results and prove useful when running classifications through a language model.

3.2. General Technique

The approach was to start with a baseline model, consisting of 3 convolutional layers with max-pooling, followed by a dense layer, and finally the output layer with 26 neurons over the softmax activation function.

The main aim was to develop a model which converges with the optimal parameters, so as to build a model which requires minimum training time and is efficient overall.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
dropout (Dropout)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout_1 (Dropout)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 128)	0
dropout_2 (Dropout)	(None, 3, 3, 128)	0
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 256)	295168
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 26)	6682

Total params: 394,522
Trainable params: 394,522
Non-trainable params: 0

Fig. 3.2.1 “Model Summary with using Dropout”

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_4 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_5 (Conv2D)	(None, 7, 7, 128)	73856
max_pooling2d_5 (MaxPooling2D)	(None, 3, 3, 128)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_2 (Dense)	(None, 256)	295168
dense_3 (Dense)	(None, 26)	6682

Total params: 394,522
Trainable params: 394,522
Non-trainable params: 0

Fig. 3.2.1 “Model Summary with using Dropout”

Dropout regularization has been used to ignore some layer outputs to prevent overfitting which was observed during training model.

3.3. Developing the pipeline

In order to obtain images of the user signing in real-time, a web application is created that is able to access a native camera on a laptop through the browser solely using HTML.

The web application sends images to the server one by one and the server classifies the image and presents the probabilities for the corresponding signs.

4. Dataset Features

4.1. Dataset Description

The dataset format is patterned to match closely with the classic MNIST. Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions). The training data (27,455 cases) and test data (7172 cases) are approximately half the size of the standard MNIST but otherwise similar with a header row of label, pixel1, pixel2, …, pixel784 which represent a single 28x28 pixel image with grayscale values between 0-255.

Below given is graph representing labels of alphabets and their corresponding number of images in the dataset.

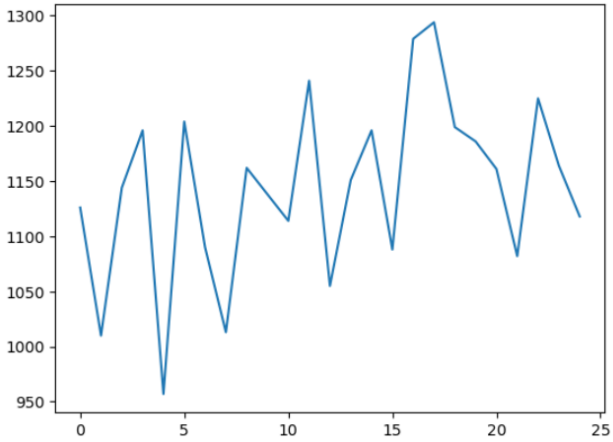

Fig. 4.1.1 Number of images to alphabets

4.2. Data Pre-processing

Both training set and test sets from the data source at Kaggle consists of image pixels as a list. So, it's required to transform the list to a (28,28,1) dimension. The pre-processing further required is separate for the model training and capturing image at from the real-time feed.

For the former, after converting the image data to its proper dimensions, the label values are given in a single separate column which are one-hot encoded using TensorFlow's to_categorical function. The pixel values range from 0 to 255 which are normalized. Data after normalization is used as input to the model.

For the latter part, frame from the video is captured using the OpenCV library. The captured frame is converted to gray and applied Gaussian Blur to smoothen the image. Further, it is resized and expand_dims are used to make it match to the required input shape of (28,28,1) which is fed to the predict method of the trained model.

5. Experiments, Results, and Analysis

5.1. Evaluation Metrics

To compare the results, the most popular criterion in the literature is used which is the validation set accuracy.

Additionally, we use a confusion matrix, which is a specific table layout that allows visualization of the performance of the classification model by class. This allows us to evaluate which letters are the most misclassified and draw insights for future improvement.

5.2. Experiments

The initial aim was to develop a CNN model which would be the simplest with optimal parameters to make it computationally efficient.

For this a baseline model was trained with three convolutional layers, each consisting of a combination of conv2d, and a max-pooling layer, followed by a dense layer and the output layer thereafter. The model was trained for 12 epochs, which showed signs of overfitting and was manually stopped.

To overcome the problem of overfitting, dropout regularization technique is used in between the layers which drop out a percentage of nodes in training. This significantly helped in training and the model converged better.

5.3. Results

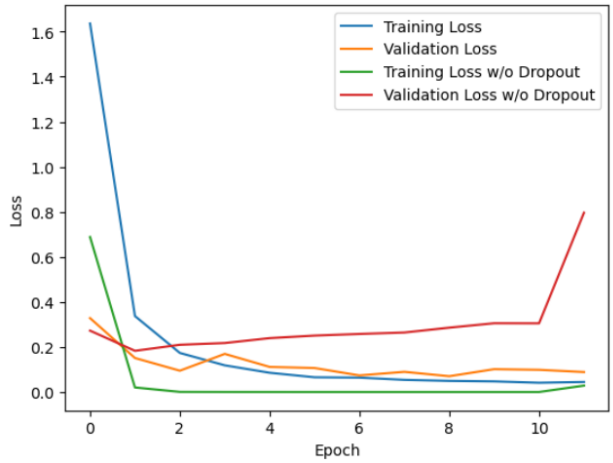

Fig. 5.3.1 “Loss”

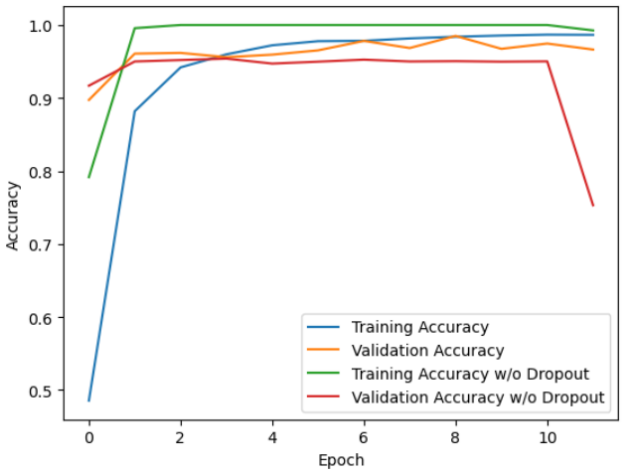

Fig. 5.3.2 “Accuracy”

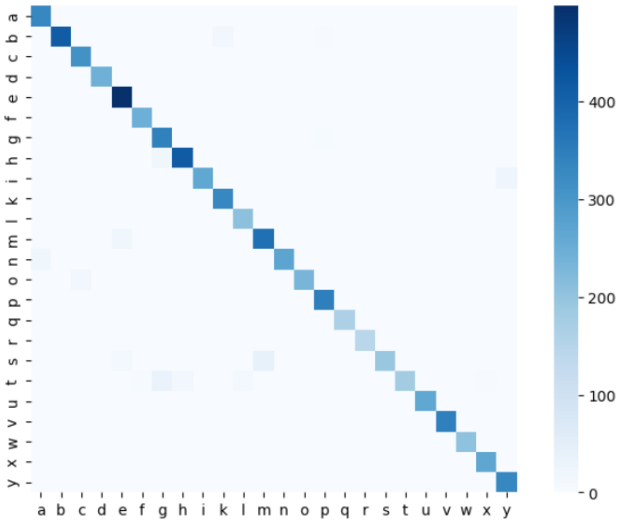

Fig. 5.3.3 “Heatmap with using Dropout”

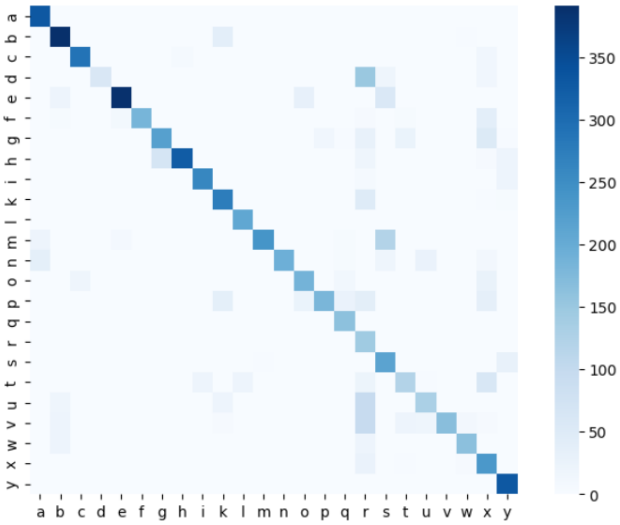

Fig. 5.3.4 “Heatmap without using Dropout”

	precision	recall	f1-score	support
A	0.94	1.00	0.97	331
B	1.00	0.96	0.98	432
C	0.96	1.00	0.98	310
D	1.00	1.00	1.00	245
E	0.94	1.00	0.97	498
F	0.98	1.00	0.99	247
G	0.86	0.99	0.92	348
H	0.97	0.95	0.96	436
I	1.00	0.92	0.96	288
K	0.96	1.00	0.98	331
L	0.95	1.00	0.97	209
M	0.90	0.95	0.93	394
N	1.00	0.93	0.96	291
O	1.00	0.94	0.97	246
P	0.98	1.00	0.99	347
Q	1.00	1.00	1.00	164
R	1.00	1.00	1.00	144
S	1.00	0.79	0.88	246
T	1.00	0.72	0.84	248
U	1.00	1.00	1.00	266
V	1.00	1.00	1.00	346
W	1.00	1.00	1.00	206
X	0.99	1.00	1.00	267
Y	0.94	1.00	0.97	332
accuracy			0.97	7172
macro avg	0.97	0.96	0.97	7172
weighted avg	0.97	0.97	0.97	7172

Fig. 5.3.5 “Classification Report with using Dropout”

	precision	recall	f1-score	support
A	0.85	1.00	0.92	331
B	0.82	0.91	0.86	432
C	0.94	0.93	0.94	310
D	1.00	0.25	0.40	245
E	0.96	0.78	0.86	498
F	1.00	0.74	0.85	247
G	0.76	0.63	0.69	348
H	0.98	0.75	0.85	436
I	0.93	0.90	0.91	288
K	0.73	0.83	0.78	331
L	0.91	1.00	0.95	209
M	0.99	0.61	0.75	394
N	1.00	0.66	0.80	291
O	0.76	0.76	0.76	246
P	0.92	0.52	0.67	347
Q	0.76	0.99	0.86	164
R	0.20	1.00	0.34	144
S	0.50	0.87	0.63	246
T	0.70	0.49	0.57	248
U	0.73	0.49	0.59	266
V	1.00	0.49	0.65	346
W	0.90	0.80	0.84	206
X	0.46	0.87	0.60	267
Y	0.81	1.00	0.89	332
accuracy			0.75	7172
macro avg	0.82	0.76	0.75	7172
weighted avg	0.84	0.75	0.76	7172

Fig. 5.3.6 “Classification Report without using Dropout”

5.4. Discussion

5.4.1 Loss and Accuracy

	Loss	Accuracy
Train	22 %	92 %
Validation	12 %	96 %
Test	8 %	96%

Table 2.4.1.1 “Metrics with using Dropout”

	Loss	Accuracy
Train	6 %	98 %
Validation	29 %	93 %
Test	79 %	75 %

Table 2.4.1.1 “Metrics without using Dropout”

From seeing the results as shown above, it can be clearly seen that the model trained without using dropout was overfitting on the training set. After adding dropout regularization to the layers in the model, the results improved significantly.

5.4.2 Confusion Matrices

The confusion matrices reveal that our accuracy suffers primarily due to the misclassification of specific letters (e.g., d and r in Fig. 5.3.4). Often the classifier gets confused between two or three similar letters or heavily prefers one of the two in such a pair (e.g., g/h in and m/n/s/t in Fig. 5.3.5).

5.4.3 Real-time user testing

Testing the model using the web application consisted of testing on the images in a wide range of environments and hands. A key observation is that there is no significant correlation between the final validation accuracy of the model and its real-time performance on the web application.

Some letters were noticeably overrepresented in the predictions. This can be attributed to the fact that neither of these has any fingers protruding from the center of the hand. They will thus share the central mass of the hand in the pictures with every other letter without having a contour that would exclude them.

6. Conclusions and Future work

6.1. Conclusion

This report documents the implementation and training an American Sign Language translator on a web application based on a CNN classifier. Because of the lack of variation in the dataset, the validation accuracies we observed during training were not directly reproducible upon testing on the web application. The hypothesis is that with additional data taken in different environmental conditions, the models would be able to generalize with considerably higher efficacy and would produce a robust model for all letters.

6.2. Future Work

The classification task could be made much simpler if there is very heavy preprocessing done on the images. This would include contrast adjustment, background subtraction and potentially cropping. A more robust approach would be to use another CNN to localize and crop the hand.

Building a bigram and trigram language model would allow to handle sentences instead of individual words. Along with this comes a need for better letter segmentation and a more seamless process for retrieving images from the user at a higher rate.

References

[1] Garcia B., Viesca S.A. Real-time American sign language recognition with convolutional neural networks. Convolutional Neural Netw. Vis. Recognit. 2016;2:225–232. [Google Scholar]

[2] Abu-Jamie, Tanseem N., and Samy S. Abu-Naser. "Classification of Sign-Language Using Deep Learning by ResNet." (2022).

[3] P. Mekala et al. Real-time Sign Language Recognition based on Neural Network Architecture. System Theory (SSST), 2011 IEEE 43rd Southeastern Symposium 14-16 March 2011.

[4] https://www.kaggle.com/datasets/datamunge/sign-language-mnist

[5] https://www.tensorflow.org/api_docs

[6] https://docs.opencv.org/4.x/

Link to the GitHub repository:
https://github.com/ronilchristian/CS584_Project