American Sign Language Interpreter

Trevor Hosek
tjhosek@iu.edu

Arinah Karim
ankarim@iu.edu

Keegan Moore
keegmoor@iu.edu

***Abstract*—This paper presents a non-invasive method to recognize American Sign Language (ASL) alphabet symbols. We inputted static images of signs into a convolutional neural network, which would then predict the symbol based on the training data. The average accuracy for symbol recognition was 95.75% for training accuracy and 98.3% for validation accuracy.**

***Index Terms*—American Sign Language, Convolution Neural Network, Computer Vision**

I. Introduction

Sign language is used worldwide and while there are existing systems that seek to translate sign language, the information may be captured unnaturally (see Background and Related Work). Therefore, our project attempts to combat this issue through the usage of neural networks, specifically CNNs as they are powerful for image processing, to correctly identify ASL alphabet symbols with an accuracy rate of at least 75%. The dataset we are using from https://www.kaggle.com/datasets/grassknoted/asl-alphabetKaggle has been cleaned and has been sorted into both training and testing sets.

II. Background and Related Work

There have been several approaches to translating sign language. The Microsoft Azure Kinect camera is capable of capturing skeletal data, such as where one's fingers are in respect to other fingers or whether or not the joints are bent[1]. There is also a hand glove that was placed on participants to determine their hand shapes during the experiment; said participant would also need to keep their hand in a certain region in order for the symbol to be detected and transcribed[2]. However, these methods do not allow the person to sign naturally.

Other related works included an analysis of a novel method for finger and gesture recognition. It firstly analyzes the hand position which it translates into a 3d map, and then extracts a gesture using a parallel edge[3]. Of note, the paper showed both real-time and offline methods for gesture extraction and recognition. Such methods would be useful for the first part of our implementation where we could recognize and classify a gesture.

Of the various works we had read, we took interest in Pigou et al.'s work with convolutional neural networks and sign language interpretation[4]. In this work, Pigou and his team implemented a CNN with the input of a Microsoft Kinect camera to identify Italian Sign Language gestures. While handling video data was a bit out of our scope, the usage of a CNN in the paper resulted in us choosing it as our methodology for tackling this issue.

III. Methodology

A. Data

The Kaggle data consisted of both training and testing data, both of which were organized into folders: a-z, space, delete, and nothing. The training data consisted of 87,000 images total. The images contained the hand of a person, and the images had various lighting and the placement of the hand in the image was inconsistent. This was useful as it provided diversity in the dataset. The testing data consisted of 29 images total, one image per symbol. Both training and testing set images were 200x200 pixels.

B. Pre-processing

Because the testing set was much smaller in size, we transferred images from the training set to the testing set such that the testing set consisted of 8700 images. Then, the training images were converted into grayscale. In order to feed the images into the CNN, an additional dimension was needed to be added to the 200x200 array to make it a 200x200x1 matrix. The pixels were then converted into floats and were rescaled to have values ranging from 0 to 1, inclusive. Because the data labels were categorical, both the training and testing image labels needed to be converted into a vector of numbers. The training images were then partitioned into a 80-20 split to reduce model overfitting.

C. Convolutional Neural Network

The keras library from Python offers modules to help build CNNs. The batch size used was 64 and the network was trained for 20 epochs. The CNN contained 3 convolutional layers, 3 max-pooling operations, a flatten layer and a dense layer and would predict what the alphabet symbol is based on the input. A dropout layer was added to help combat overfitting the model.

We used Leaky ReLU for the activation function as it prevents the death of ReLU units. This results in the weights of the network to not drastically change, which helps prevent the death of those units.

After the construction of the model, we used the Adam optimizer instead of a classic stochastic gradient descent procedure to update the neural network weights. We then applied the categorical cross entropy loss function as we had multi-labels. First, the input was passed into a 32 3x3 filters to extract a feature map. The data was then put into the Leaky ReLU.

Next, a 2x2 max-pooling was applied to reduce overfitting. A dropout layer with a rate of 0.25 was added to also help with overfitting. Following that was an additional convolution layer with 64 3x3 filters and followed the same steps as above. The data was then fed into the last convolutional layer which consisted of 128 3x3 filters. It also followed the same flow as above, except the dropout rate became 0.4 instead of 0.25. The model was then flattened, passed through a linear dense function, inputted into another Leaky ReLU function where it then entered another dropout layer, and finally softmax was applied to guess the correct label. Our model was adapted from Data Camp's tutorial on CNNs[5].

IV. Results

The table depicts the results of training for 20 epochs. Between 19 and 20 epochs, the different losses and accuracies began to change only slightly. However, the validation accuracy is much higher than the accuracy of the model, which means that the model is definitely overfitting the data.

Epoch	Loss	Accuracy	Val$_loss$	Val$_accuracy$
16/20	0.1500	0.9482	0.0714	0.9783
17/20	0.1573	0.9458	0.0586	0.9835
18/20	0.1405	0.9515	0.0475	0.9855
19/20	0.1304	0.9573	0.0554	0.9823
20/20	0.1297	0.9575	0.0578	0.9830

This means that the model works well on the data it is given, but is not guaranteed to work well on new testing data. Figure 1 depicts how the model began to memorize the data around 2 epochs as the validation accuracy started to remain around the same from there on. Even with the help of the dropout layer, overfitting was still an issue to be solved. The model

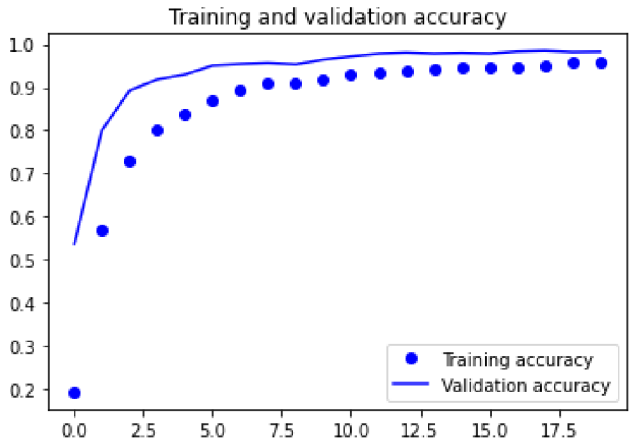

Fig. 1. CNN Overfitting

produced a 0.0558 loss with a 98.24% accuracy on the testing data. Of the 8700 predicted labels for the testing images, 8503 of the images had been correctly labeled. Figure 2 depicts the input image and the label it was given. Figure 3 shows examples of cases where the labels were incorrect on the testing images. There may be some underlying relationship

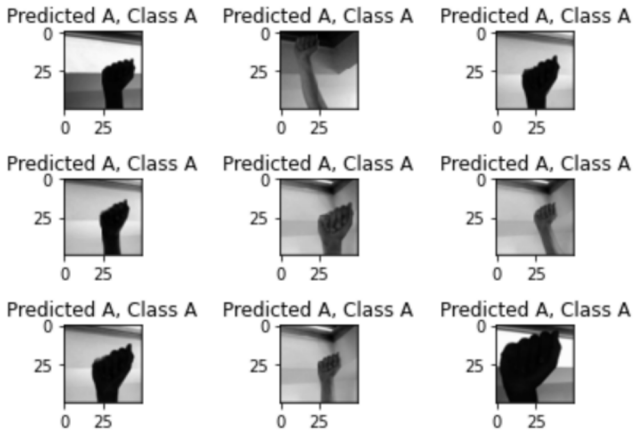

Fig. 2. Correct Labeling

between the predicted and actual label for each of the wrongly labeled images.

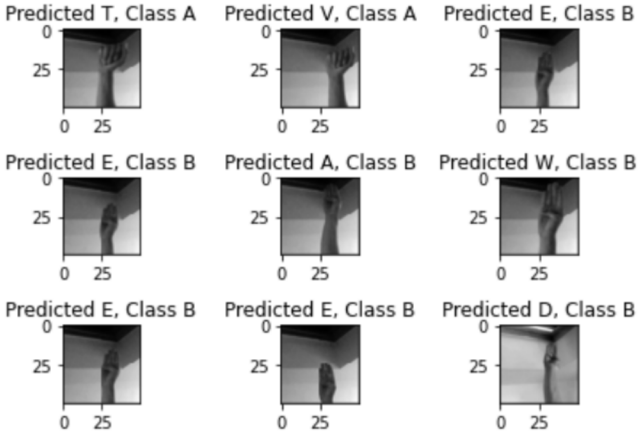

Fig. 3. Incorrect Labeling

For each of the labels, the precision, recall, f-1 score, and support were calculated. The support was a constant 300, as there were 300 images per label for the testing set. The table below shows the results of each of the labels.

Class	Recall	Precision	f-1 Score
A	0.86	0.99	0.92
B	0.99	0.97	0.98
C	1.00	1.00	1.00
D	0.98	1.00	0.99
E	0.99	0.94	0.96
F	1.00	1.00	1.00
G	1.00	0.99	0.99
H	0.99	0.99	0.99
I	0.99	0.99	0.99
J	0.99	1.00	1.00
K	0.98	0.99	0.99
L	1.00	1.00	1.00
M	1.00	0.98	0.99
N	0.98	1.00	0.99
O	1.00	0.96	0.98
P	1.00	1.00	1.00
Q	1.00	1.00	1.00
R	1.00	0.95	0.97
S	0.98	0.97	0.97
T	0.98	0.98	0.98
U	0.96	0.89	0.92
V	0.95	0.92	0.94
W	0.93	1.00	0.96
X	0.97	0.96	0.97
Y	0.98	1.00	0.99
Z	0.97	1.00	0.98
Del	1.00	1.00	1.00
Nothing	1.00	1.00	1.00
Space	1.00	1.00	1.00
accuracy			0.98
macro avg	0.98	0.98	0.98
weighted avg	0.98	0.98	0.98

Based on the table, A had the lowest recall rate and U had a much lower precision rate, which means that it had much more true negatives than the others. C, F, G, L, M, O, P, Q, R, delete, nothing, and space had a rate of 1.00 for recall, which means it was able to identify all of the true positives in the testing set.

V. Discussion

While the model seemed to perform very well, there was an obvious issue with it. The model succeeded in memorizing the training data and because the testing set consisted of images very similar to the training set, this probably resulted in the great accuracy that we saw. A solution to this is to go through the training set and remove redundant images and reduce the size of the training set by a greater factor. Additionally, the images could have been transformed to help diversify the training set.

Realistically, when someone is signing a language, it will be in the form of a video. While this approach uses static images and produces a decent accuracy, analyzing a video will be a harder problem to tackle. However, this is a good start to our research question of finding a non-invasive method to help identify American sign language symbols.

VI. Conclusion

The program produced a much higher accuracy than what we set our baseline as. In the future, we could try to correct the overfitting of the model. Additionally, we could try to take a picture of a person and find the hand and then have the CNN guess the symbol made. Another step to advance this project is to work with videos and decipher the gestures in the video. That would be a challenging task, but this project is a step closer towards it. This project shows that there are non-invasive methods to solving problems such as these and that they can perform just as well as the methods that do use invasive technology.

VII. References

[1] Microsoft, ”Kinect Sign Language Translator”, 2013.

[2] Zhou Y., Jiang G., & Yaorong L., “A novel finger and hand pose estimation technique for real-time hand gesture recognition,” 2015.

[3] Kulkarni V., Lokhande S.D., “Appearance Based Recognition of American Sign Language Using Gesture Segmentation,” 2010.

[4] Pigou L. et al., “Sign Language Recognition Using Convolutional Neural Networks,” 2014.

[5] Sharma, ”Convolutional Neural Networks in Python with Keras”, 2017.