American Sign Language with Convolutional Neural Networks

Addison Boyer
Machine Learning (CSCI 547)
Fall 2019

1.0 Introduction

This section provides of an overview of American Sign Language, as well as various computational techniques for Sign Language Recognition.

1.1 American Sign Language

According to The National Institute on Deafness and Other Communication Disorders (NIDCD) American Sign Language (ASL) "is a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English." As with most languages, ways of expressing certain ideas in ASL vary from person to person. For example, within the American Sign Language there exists variations in signing rhythm and pronunciation as well as slang and even the actual signs that are used[**2**]. Although ASL, and more generally sign languages certainly vary from region to region, they tend to share two key characteristics. Namely, sign languages consist of manual and non-manual signing.

Figure 1.1.1 the ASL alphabet.

Manual signs consist of the positioning of the hands, as well as their orientation, trajectory, and shape. For example, Figure 1.1.1 shows the manual signs for the alphabet in ASL. In contrast, non-manual signing consists of the body language, and facial expressions used in conjunction with manual signing. This component is often helpful to emphasize, or convey the meaning of the manual signs. However, the manual signs tend to carry the majority of the important lingual information[**3**]. With this in mind, researchers generally chose to focus their Sign Language Recognition (SLR) studies on the former.

1.2 Sign Language Recognition

Sign Language Recognition is the process of automating the translation of sign language by means of machine learning and/or computer vision techniques. Without automated forms of SLR, a human interpreter is almost certainly required. For these very reasons, automated forms of SLR have become extremely important. Most modern forms of SLR use either vision, or sensor based techniques. Vision based techniques often cost more computationally, but have the added benefit of not requiring the person signing to wear extra equipment[**3**]. Vision based techniques generally only require a camera, a device that anyone with a modern cell phone certainly has. These types of techniques will be the main focus of the work outlined in this paper.

In this paper I aim to create an ASL sign language recognition system using a Convolutional Neural Network that takes gray-scale images as input, and outputs a classification of those image (0-25 or A-F). The remainder of this paper is organized into 5 sections consisting of a literature review, methods, results, discussion, and conclusion.

2.0 Literature Review

This section presents a few examples of works related to sign language recognition using vision-based techniques. The methods used, as well as the results of the studies are discussed below.

2.1 SIFT Keypoint Matching

In [**4**], the authors have introduced a sign language classification algorithm that utilizes SIFT keypoints for classification. Their software pipeline consists of four steps; Image Acquisition, Feature Extraction, Orientation Detection, and Gesture Recognition.

Images are acquired through a Matlab camera application, and then saved to a directory for later Feature Extraction. During the feature extraction step, the images are transformed into large collections of local feature vectors. Then, a series of filtering operations including scale space extrema detection and Keypoint localization are applied to the images. This filtering reduces the total number of keypoints to process in the remaining steps.

Next, a consistent orientation is assigned to each Keypoint using local gradient data, and a

binning (histogram) approach. Using this data, Keypoint descriptors are identified in the image. Lastly, keypoints are calculated for each image in the database, and matched against those of the input image. The authors of this study concluded that their algorithm did in fact work, and could be used to decode the frames of sign language videos.

2.2 Convolutional Neural Networks

In [**1**], the authors have introduced a sign language classification algorithm that utilizes a Convolutional Neural Network for classification. The network consists of four main parts; The input layer, convolution and pooling layers, and the classification layer.

The input to the CNN is an image of a finger sign in ASL consisting of three 32x32 feature maps, and a single 32x32 feature map for the depth. Next, these feature maps are transformed by a series of convolutions, and pooling operations. Finally, these feature maps are flattened and passed into a fully connected feed forward neural network for classification. This network consists of 2 layers, one hidden layer (128 nodes), and an output layer consisting of 24 nodes (one for each class).

Their network was tested on a data set of 60,000 images (500 for each sign) and excluded J and Z due to their need for motion. The experiment was run for approximately 250 epochs, or until their network was fully converged. The ConvNet architecture that they had produced was able to achieve nearly 82% precision, and 80% recall.

3.0 Methods

As previously mentioned, the goal of this work is to create an ASL sign language recognition system that utilizes a Convolutional Neural Network for classification. The first step in solving this problem was to identify a robust data set to train the network.

3.1 MNIST Data set for ASL

The original MNIST data set of handwritten digits is a popular benchmark for image-based machine learning methods. Researchers have attempted to enhance this dataset and/or provide drop-in replacements which have real world applications. Among these replacements is the MNIST data set for American Sign Language.

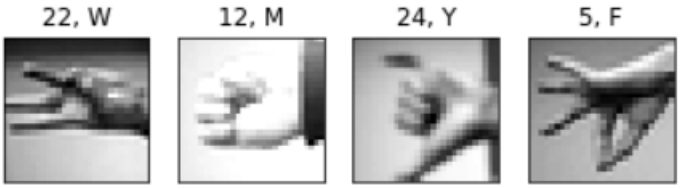

Figure 3.1.1 A small subset of the MNIST Data set for ASL.

Initially, the data set consisted of 1,704 color images. The data set has since been augmented to increase the quantity of data. To accomplish this, a number of transformations were applied to the original images including gray-scaling, cropping around the hand, resizing, filtering, random pixelation, and rotation. The resulting data set consists of 34,627 gray-scale images, each accompanied by a label identifying it's letter representation. A small subset of the augmented data set is shown in Figure 3.1.1 above.

3.2 Data Preparation

The format of the data set closely resembles classic MNIST. Each training, and test image has a label (0-24) which maps to a letter (A-Z). There are no labeled images of J=9 or Z=24, as they require gesturing motions beyond the scope of this paper. The label mappings for the images are shown below in Figure 3.2.1.

Class	Letter
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
9	J
10	K
11	L
12	M
13	N
14	O
15	P
16	Q
17	R
18	S
19	T
20	U
21	V
22	W
23	X
24	Y
25	Z

Figure 3.2.1 Letter to Class Mappings.

Each label and image is read in from a .csv file whose rows consist of a single label followed by 784 gray-scale pixel values between 0 and 255. These pixel values of course make up the 1x28x28 images that will soon become the input to our Convolutional Neural Network. The data set is separated into training and test cases. The size of the training data set is 27,455 whereas the size of the test data set used to validate the model is 7,172. The images comprising the training and test sets were pre-determined, and are not ran-

domly chosen each training session.

3.3 Network Architecture

The proposed model is implemented in PyTorch, a machine learning package for Python. The model takes an array of 1x28x28 images as input. The input is then passed into a series of 3 convolutional layer blocks. In each convolutional layer block the feature map(s) are first convolved with a kernel of size 3, and with a padding of 1. After each convolution a non-linear activation function is applied to each feature map. This activation function is ReLU or Rectified Linear Unit. The equation for the rectified linear unit is shown below.

$$ReLU(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

Equation 3.3.1 Rectified Linear Unit (ReLU).

Finally, after ReLU activation, each feature map is down sampled via a max pooling operation with a kernel size of 2, and a padding of size 2. The output is then passed as input to the next convolutional layer block. After the series of convolutional layer blocks the output is flattened and passed into the fully connected feed forward layer as input.

In the fully connected feed forward layer a series of of linear transformations and ReLU activation's transform the output from the convolutional layer into logits, or probabilities of class membership. In each hidden layer, the size of the features is nearly halved, and a dropout of approximately 10% is applied the nodes of that layer. Adding dropout to the model makes it more robust, and helps the model to not be over fitted.

3.4 Model Training

Training the model correctly requires the minimization of a cost function with respect to the model's parameters. The loss function that this model will use is Categorical Cross Entropy loss. The loss function is used in conjunction with gradient descent in order to adjust the parameters (via back propagation) and train the model. A batch size of 256 is utilized, and the model is trained for 15 epochs. Due to the complexity of the network, it is recommended to train the model on a GPU. Training on a CPU takes approximately 90-120 minutes.

4.0 Results

4.1 Training and Test Set Accuracy

After training for 15 epochs the model was able to achieve approximately 100% Training set accuracy, and 94% Test set accuracy. Training and Test set accuracy by epoch can be found in Figure 4.1.1 below.

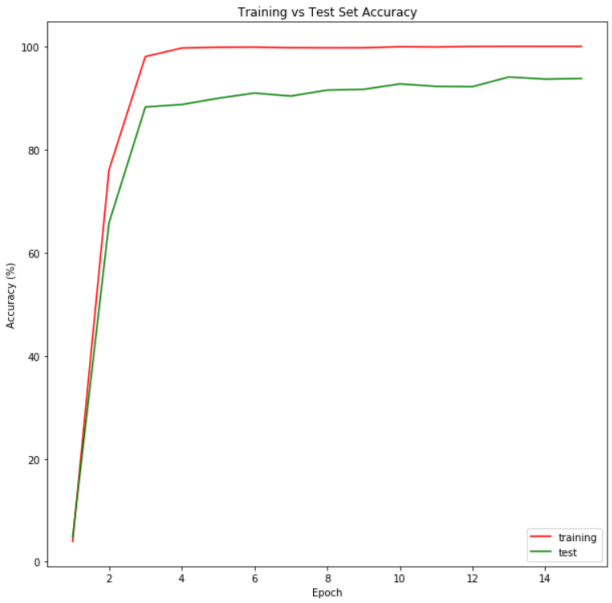

Figure 4.1.1 Training and Test Set Accuracy per Epoch.

4.2 Convergence of the Model

The model appears to have converged, as the loss is tending towards 0. This can be seen in Figure 4.2.1 below.

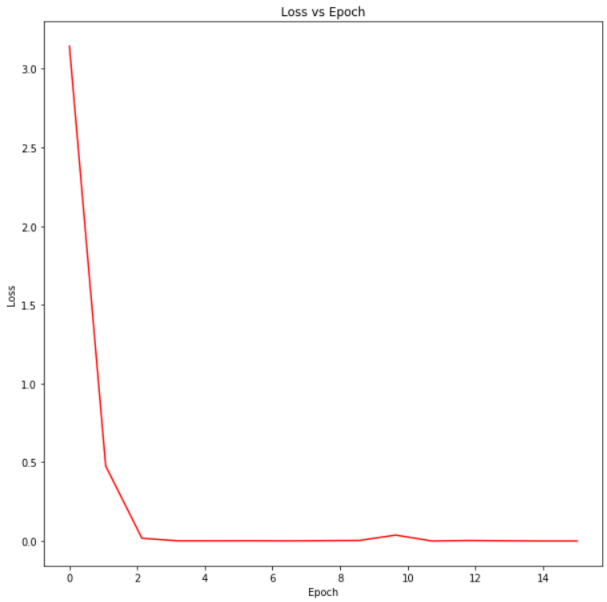

Figure 4.2.1 Categorical Cross Entropy Loss per Epoch.

4.3 Precision and Recall

In order to learn what the model was good at classifying, as well as what it struggled with a

confusion matrix was generated. A graphical representation of the confusion matrix can be found below in Figure 4.3.1.

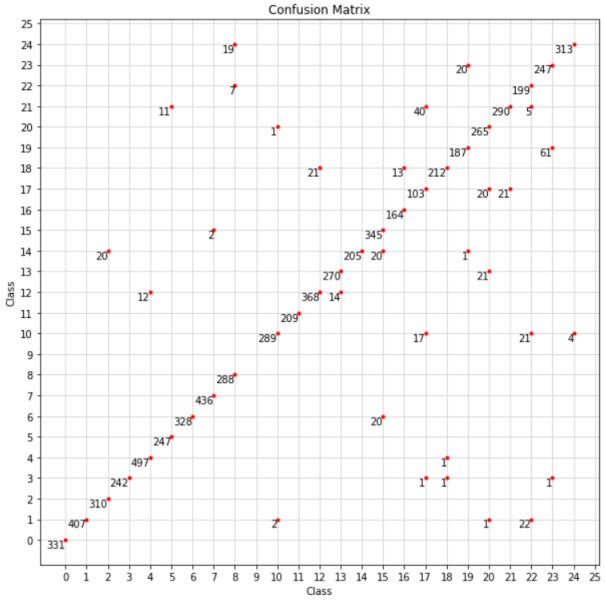

Figure 4.3.1 The Confusion Matrix of the Trained Model.

Precision	Recall	Letter
100.00	100.00	A
100.00	94.21	B
93.94	100.00	C
100.00	98.78	D
97.64	99.80	E
95.74	100.00	F
100.00	94.25	G
99.54	100.00	H
91.72	100.00	I
98.97	87.31	K
100.00	100.00	L
94.60	93.40	M
95.07	92.78	N
100.00	83.33	O
89.61	99.42	P
92.66	100.00	Q
63.98	71.53	R
99.07	86.18	S
89.90	75.40	T
86.32	99.62	U
93.25	83.82	V
80.57	96.60	W
79.94	92.51	X
98.74	94.28	Y

Figure 4.3.2 Precision and Recall by Letter.

Precision and Recall are good metrics to measure how well the model is performing on certain classifications. Precision is defined as the total number of correctly classified positive examples, over the total number of positives. In contrast, Recall is the total number of correctly classified examples, over the total positives. Precision and Recall calculations are displayed in Figure 4.3.2 and 4.3.3.

```
Avergae Precision: 93.39
Avergae Recall: 93.47
```

Figure 4.3.3 Average Precision and Recall for the Trained Model.

5.0 Discussion

5.1 Confused Images

A confusion matrix like the one in Figure 4.3.1 is a great tool to identify what images the CNN had difficulty classifying, and which images it classified with ease. The class on the x-axis is the predicted class, and the class on the y-axis is the actual class. The model was able to achieve an average precision of 93.39%, and an average recall of 93.47% as seen in Figure 4.3.3 above.

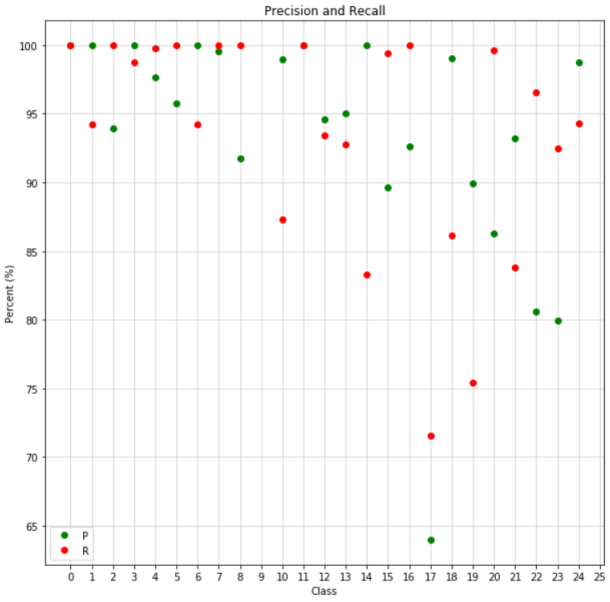

Figure 5.1.1 A Graphical Representation of Precision and Recall from Figure 4.3.2.

With this in mind, it's useful to examine the classes that performed drastically worst than the average in either category. These classes can be easily identified in Figure 4.3.2 above in conjunction with the confusion matrix in Figure 4.3.1. It appears classes 17 and 19 have the lowest recall and/or precision of all the other classes.

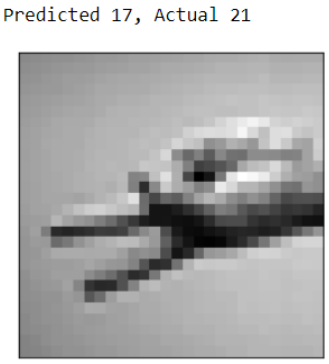

Figure 5.1.2 An image predicted to be an R, but is actually a V.

According to the confusion matrix there are approximately 40 images that were classified as R's but were actually V's. An example can be found in Figure 5.1.2 above. It appears that the model learned the lower hand positions, and not the split fingers that distinguish the two from each other. Observe that this is the main difference between the two letters in Figure 1.1.1.

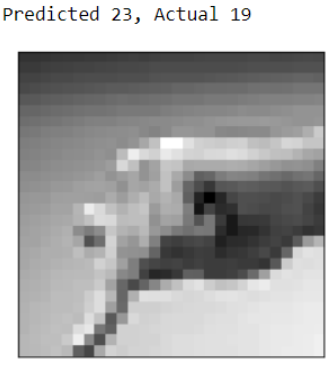

Figure 5.1.3 An image predicted to be an X, but is actually a T.

According to the confusion matrix there are approximately 61 images that were classified as X's but were actually T's. An example can be found in Figure 5.1.3. It appears that the model did not learn how to distinguish the position of the pointer finger in each of these signs (see Figure 1.1.1).

5.2 Study Comparison

Compared to the other studies in the literature review section, this model was most similar to that described in [**1**]. One key difference is that their network takes 4 feature maps as input, whereas our model takes simply one. Their model was able to achieve around 82% recall, and precision. This is one area in which our model seems to outperform theirs.

In their study the model was trained for hundreds of epochs, whereas ours was only trained for 15. Our model architecture is more complex than theirs, and utilizes a different activation function. They used a Leaky ReLU activation function, where the model outlined in this paper used regular ReLU. Our model also utilized Dropout, which may have contributed to a higher test set accuracy. It appears that our model has converged, but potentially it could've been trained for more epochs for an increase in test set accuracy.

6.0 Conclusion

In conclusion, the proposed model architecture was able to produce 100% test set accuracy, and 94% training accuracy. This model appears to have out-preformed the models outlined in various other research studies using Convolutional Neural Networks as a Sign Language Classifier. It appears that our model had some difficulty classifying signs that were extremely similar, namely T, X, R, and V.

It appears the model might have benefited from a longer training time (i.e. more epochs). The model was implemented in Python, and the code base can be found in AddisonCodeBase.py. Here, all elements of reading in the data, training the model, and validating the model have been implemented.

6.1 Future Work

Now that an accurate model has been trained for ASL sign language classification, it can be applied to the problem of Sign Language Recognition. The next step would be to build an application using this trained model, that could interpret sign language from the user. The start of such application can be found in Webcam.py. A screenshot of the initial prototype can be found in Figure 6.1.1 below.

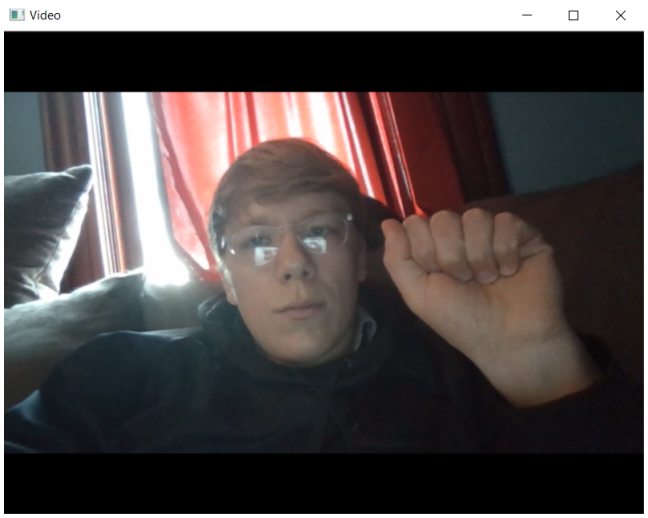

Figure 6.1.1 A prototype interface for a SLR

system using the model outlined in this paper.

References

[**1**] Ameen, Salem. and Vadera, Sanil., "A convolutional neural network to classify American Sign Language fingerspelling from depth and colour images", *WILEY Expert Systems*, 2016, pp 1-8

[**2**] "American Sign Language." National Institute of Deafness and Other Communication Disorders, U.S. Department of Health and Human Services, 8 May 2019, www.nidcd.nih.gov/health/american-sign-language.

[**3**] Benaddy, Mohamed., El Meslouhi, Othmane., and Hayani, Salma.,"Arab Sign language Recognition with Convolutional Neural Networks", *IEEEICCSRE2019*, 2019, pp 1-4

[**4**] Goyal, Sakshi., Sharma, Ishita., and Sharma, Shanu., "Sign Language Recognition System for Deaf and Dumb people", *International Journal of Engineering Research & Technology*, 2013, pp 382-387