2021 CSE 40626 Final Project Report

Prepared by

Carter Goldman, Livia Johan, Christina Youn

University of Notre Dame,

Notre Dame, IN, 46556

1. Objectives and Expected Significance

We often see sign language interpreters translating speech into sign language, but we don't often see communication going the other way. Through our project, we hoped to be able to help improve the lack of this directional communication. Therefore, we wanted to create an image recognition model that can recognize 24 letters in the American Sign Language alphabet (letters excluding J and Z because they require motion). We planned to be able to feed in a live video stream and have the trained model predict the letters in real-time by running the model on each frame.

2. Related Background

The dataset we used can be found here. This dataset contains images from "multiple users" (we do not know exactly how many) and was expanded and modified through the use of "filters ('Mitchell', 'Robidoux', 'Catrom', 'Spline', 'Hermite'), along with 5% random pixelation, +/- 15% brightness/contrast, and finally 3 degrees rotation" according to the documentation.

It is formatted as a CSV file where each row has 785 columns; the first column in each row is a categorical label between 0-25 (representing the letters A-Z), and the following 784 columns are integers between 0-255 (representing a grayscale pixel in a 28 x 28 image). However, there are no images for *J* (9) or *Z* (25) because these letters require movement. When we filtered through this dataset, we eliminated the vacancies at 9 and 25 so that their absence would not introduce any bias or error. As a result, K became 9, L became 10, and so on. All design and project references we used came from our own previous assignments and experience in this and prior classes. We did not use any outside sources aside from the library documentation provided by Tensorflow and OpenCV.

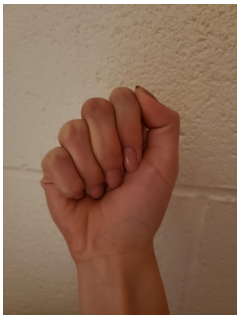
3. Approach

We took a two-step approach to training and testing the model.

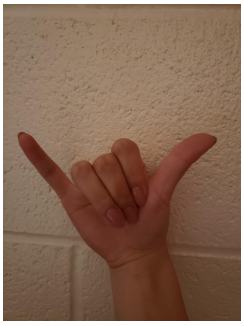
The first step was training the model. This involved loading in the data and preprocessing the data for use within the model. We removed the vacancies at 9 and 25 for *J* and *Z* respectively, reshaping the input data into an n x 28 x 28 x 1 array (where *n* is the number of images in the dataset) and formatting the labels with one-hot encoding. We then fed this into our finalized neural network with two convolutional layers (the first with 128 neurons and a 5 x 5 kernel, the second with 128 neurons and a 3 x 3 kernel, each with a selu activation function) with 2 x 2 two-dimensional max pooling layers in between to reduce the variance. The last layers of the network were a flatten layer to reduce the multi-dimensional matrix into one-dimensional and a

dense layer with 24 neurons and a softmax activation function as the output layer. The model used categorical cross entropy for the loss function, adam as the optimizer, and accuracy as the performance metric. We trained the model for eight epochs with a batch size of 100, and used 20% of the training data for validation.

The second step was testing the model. Our first test was with the dataset's provided testing data. Then, we wanted to test the model with our own photographs. Here are the photographs we took:



a.jpg



y.jpg

To test these photos, we had to do some initial image preprocessing to match our training images. This meant that we had to (1) recolor, (2) crop, and (3) resize each image. We accomplished each of these steps by using OpenCV. When recoloring the images, we used the *COLOR_BGR2GRAY* filter to change our images to black and white. We then cropped the images to be squares by taking the middle of the screen: the left and right panes of landscape images were removed while the top and bottom panes of portrait images were removed. Finally, each image was compressed to be 28 x 28 pixels. After doing these image preprocessing steps, we loaded our trained model and tested the model on our own images.

4. Project Outcome

We were able to create a machine learning model that takes input from a live video stream. Our program predicts letters in real-time by running the trained model on each frame of the video. This prediction is printed on the console. Here is a video that demonstrates our final project. The command to run our program is: `python project.py`

5. Unexpected Events

We did not experience any unexpected events. We were able to accomplish our project according to the proposal in a timely manner.

6. Lessons Learned

We learned that data preprocessing and replication of training data is of great importance to performance of a machine learning model. Our model predicted the letter shown within the dataset 95% of the time, but was seemingly much worse than 95% on our live video feed. However, every image in the dataset was taken of hands in the same lighting and environmental conditions. In order to achieve greater generalization of the model, either our training dataset would have had to have had a greater scope (different people, different backgrounds, different lighting conditions, different camera angles, different hand positions), or we would have had to preprocess our own testing data more to be much more like the original dataset (lighting correction, feature detection and cropping, etc).

In trying to create a more accurate model, we created multiple iterations of different parameters. We first drew from our work in one of the mini-projects, where we found that selu, relu, and elu rendered some of the most accurate models in those assignments. As a result, we also implemented them in our final project. Moreover, we learned that when we increased the number of neurons beyond 128, two things occurred: (1) the model would overfit and perform worse on the test data on average and (2) it would take far longer to train models with this many neurons, and the payoff, if applicable, was often very marginal. The accuracy from these models would be in the 80s, which was higher than some of the other models we created, but they weren't the most accurate.

7. Project Performance

For the first variable of on-time performance, we hit every milestone we set out to achieve on time. By the first milestone on 4/23, we set out to write code to preprocess our training data and have an initial convolutional neural network model created, both of which we achieved. By the second milestone on 5/7, we set out to have the model structure finalized, test the model on our own photos, and start attempting to feed in a video stream, all of which we achieved. We achieved everything we set out to do by the final project deliverable.

As for the original expectations, we set out to create a model that can identify American Sign Language letters in a live video stream. Our final project deliverable achieves this goal to a satisfactory degree, but there is room for improvement. The model we created predicts the correct sign language letter in the training dataset's testing data with over **95% accuracy**; however, with data from the live video stream, the model performs somewhat worse, as the environmental conditions are different from the dataset we trained on. However, the model is still decently accurate at predicting sign language from the live video feed.

8. Individual Contributions (40625 only)

All three members contributed equally to the success of this project. Prior to working on the coding portion of this project, we met as a group to research potential projects that we could work on. Once we began working with the data and the models, the breakdown of the work we did is as follows:

Livia & Carter worked together on:

- Save the dataset from the website to Google Drive, then load it into the Python module from Google Drive
- Sift through the data and eliminated the vacancies left by J and Z
- Plot the compressed black and white images based on the data (not raw images) that were given to us by the dataset
- Made many attempts to create a working model that would train and test successfully on the dataset that we were given. For our first status report, we were able to create an initial model with 82% accuracy. After adjusting the parameters multiple times, we were able to render a model that had a 95% accuracy, which we used in testing images that we took ourselves and in the livestream

Christina:

- Optimized and organized initial code
- When Carter and Livia ran into problems with the shape of the dataset not fitting into the expected parameters of training and testing the model, Christina was able to fit it into the correct shape. As a result, our initial model was able to run successfully.
- As Carter and Livia were working on optimizing the accuracy of the model, Christina wrote code using the OpenCV module so that we could have a live video input stream
    - The code that needed to be written for the live input stream was not dependent on using the final rendition of the model, which is why the two tasks were able to be done simultaneously
- On the video stream (which is visible to the viewer live), Christina set square boundaries within which the image we use for testing would be located. This region in the video feed was then recolored into black and white and then compressed so that it would be 28 pixels x 28 pixels, which is in line with how the dataset represented the testing images.