American Sign Language Classifier

Introduction & Research

Over 5% of the world's population—or 466 million people—suffer from disabling hearing loss. It is estimated that by 2050, one in every ten people will have disabling hearing loss. This includes one-third of the population over 65 years of age. I wanted to develop an app that could 'read' American Sign Language (ASL) so that people who didn't understand ASL could communicate with those with hearing loss. My proposed sign language classifier uses static sign language images and a Convolutional Neural Network (CNN) to identify the signs represented by each of these images. The goal was to build a system that could correctly identify American Sign Language signs that correspond to hand gestures to enable the deaf and hard-of-hearing community to communicate better with members of the hearing community.

For instance, there have been incidents where those who are deaf have had trouble communicating with first responders when in need. Although first responders and medical personnel may receive training on the basics of ASL, it is unrealistic (as of yet) to expect everyone to gain complete fluency in sign language. Previous research has attempted to create such a system, but either had an incomplete alphabet set or had trouble with existing image databases, rendering recognition incomplete. If successful, this computer vision/recognition program could aid a first responder in understanding and helping those who are unable to communicate through speech. Additionally, there are people who can translate American Sign Language in real time, but these translators are expensive and often not available.

Machine learning is a concept that has existed since the 1950s, when various mathematicians and psychologists began to experiment with the idea of modeling brain cell interactions. In the past decade, the concept of machine learning and neural networks has become much more feasible with quick to implement libraries from Keras and Tensorflow. When doing supervised learning, Keras takes the pixel-by-pixel information of the image and uses it to create an algorithm to classify. With an input layer, output layer, and any given number of hidden layers (each with 2^n neurons), the model pulls features from the data and uses it to build weights. With each passing of the data, the model tries different sets of weights on each neuron to decrease the loss.

Hypothesis

The overarching goal was to build a system that could correctly identify ASL that corresponds to hand gestures contained in static images. The proposed performance and model for the still images was to be able to identify the static signs with an accuracy of 95%, which would be in line with previous research. A predicted issue would be that there would be problems with classifying the letters j and z, as they require movement and that is almost impossible to classify with static images.

Materials

- Laptop (2013 Macbook Pro)
- Tensorflow 2.1.0
- Keras
  - Conv2D
  - MaxPool2D
  - Flatten
  - Dense
  - Dropout
  - Sequential
- Python 3.7.3
- Pip 19.0.3
- Numpy
- Pandas (library)
- Jupyter Notebook

Procedures

Using the publicly available Sign Language MNIST dataset from Kaggle, I created models to classify hand gestures for each letter of the alphabet. Due to the motion involved in the letters j and z, these letters were not included in the dataset (this is a function of the Kaggle dataset, not my system). However, the remaining dataset includes approximately 25,000 28x28 pixel images of the remaining 24 letters of the alphabet. Similar to the original MNIST hand drawn images, the data contains an array of grayscale values for the 784 pixels in each image.

Training was done by using a convolutional neural network (CNN) model to classify the static images in the dataset. The first goal in building the neural network was to define the input layer. A 28x28 image contains 784 pixels each represented by a grayscale value ranging from 0 (black) to 255 (white). By converting each image to a series of numbers, the data was transformed into a computer-readable format. The MNIST-sized images (28x28 pixels) helped train the model faster, Keras and tensorflow machine learning libraries were installed. Images were then organized into pandas dataframes, a way of storing the image that the keras library would recognize. Then, with the help of the library, the data frame was flattened to input in the model and use the CNN to train it. Once the input layer was prepared, it could be processed by the neural network's hidden layers. To increase the accuracy, I changed the number of layers, neurons per layer, MaxPools, Dropouts, and epochs. For an individual independent variable, I found the number that worked best then changed the next variable. Two factors were never changed at once.

Epochs	Function	Train Acc.	Test Acc.	CNN layers	Dense layers	MaxPools	Dropouts	Optimizer	Loss Function
10	ReLU	97.25	68	N/A	(128)	0	0	Adam	Sparse
12	ReLU	88.30	80	N/A	(128)	0	0	Adam	Sparse
12	tanh	98.94	76	N/A	(128)	0	0	Adam	Sparse
12	Sigmoid	96.70	75	N/A	(128)	0	0	Adam	Sparse
12	Linear	99.20	68	N/A	(128)	0	0	Adam	Sparse
12	Softmax	92.15	75	N/A	(128)	0	0	Adam	Sparse
12	ReLU	98.30	89	(32,32,64)	(64)	1	1	Adam	Sparse
12	ReLU	99.60	91	(32,32,64)	(64)	1	1	Adam	Sparse
10	ReLU	99.87	80	(32,32,64)	(64)	0	1	Adam	Sparse
10	ReLU	99.89	89	(32,64,128)	(64)	1	2	Adam	Sparse
10	Sigmoid	6.66	2	(32,64,128)	(128)	2	1	Adam	Sparse
8	ReLU	98.37	93	(32,64,128)	(128)	1	3	Adam	Sparse
8	ReLU	97.69	90	(32,64,128)	(128)	1	3	Adam	Sparse
10	ReLU	99.37	95.9	(32,64,128)	(128)	2	3	Adam	Sparse
12	ReLU	99.31	96.8	(32,64,128)	(128)	2	3	Adam	Sparse
10	ReLU	99.41	95.0	(32,64,128)	(128)	2	3	Adam	Sparse
12	tanh	99.51	85.8	(32,64,128)	(128)	2	3	Adam	Sparse

Figure 6: This graph shows variations of models created to increase the test accuracy. The final model with the highest accuracy is the third row from the bottom.

Figure 4: This is a set of predictions that the model has made on images it has never seen before. The blue bar represents a correct classification and its height represents its confidence.

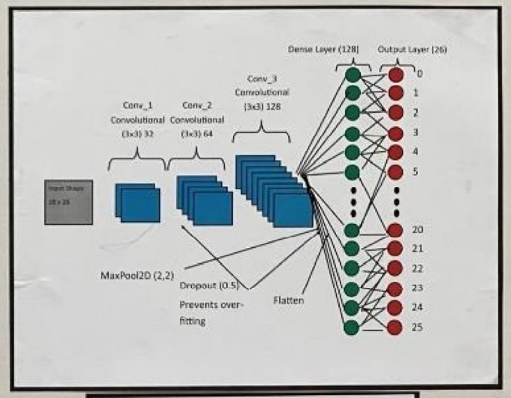

Figure 3: This is a diagram showing the structure of the convolutional model that reached the highest test accuracy (96.8%).

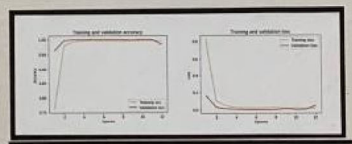

Figure 8: These are graphs representing the loss and accuracy verus epoch for validation and training in the model that tested at 96.8 percent accuracy

Data Interpretation

My first try at a model to recognize American Sign Language was a simple sequential Keras model with only one hidden 128 neuron layer. While the training accuracy reached 97.5%, it didn't classify as well on the test data (reaching only 68 percent accuracy). In a real-world application, recognizing just over two-thirds of signs correctly would not suffice, so I wanted to increase accuracy. I tried linear, sigmoid, tanh, and softmax activation functions for that hidden layer and was only able to reach 76 percent test accuracy. While I had seen people classify handwritten digits with high accuracy with only one simple dense layer, I realized that images of ASL are much more complicated. Many signed letters are extremely close in shape with only slight differences in finger position. Thus, I needed a convolutional neural network that could extract features better than a single dense layer. When I trained a model with three convolutional layers (32,32,64) and one dense layer (64), I got a train accuracy of 99.87 and a test accuracy of 80 percent. This suggests extreme overfitting of the data. So, after experimenting with a combination of maxpools and dropouts, I finally found a model that still overfits the data but only does so by 2.5 percent (99.3% train and 96.8% test).

In models where data has been overfit, the train accuracy is much higher than the test accuracy. The train accuracy is found through testing the model on the data it used to train itself. Test accuracy is more realistic as it shows how the model performs with unseen data.

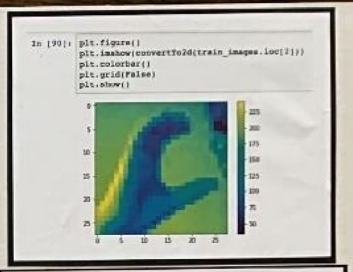

Figure 1: This image shows a 28 by 28 pixel image of the letter "C" in American Sign Language. Each pixel is a value between 0 and 255.

Future Work

The next step would be moving toward real time recognition of images. There is an open source computer vision library called OpenCV that allows models to be deployed with live classification. It allows the program to use a laptop's webcam to capture the frames of a video and run it through the model. Additionally, to simplify the model and remove background distractions, there is a method called skin detection that takes a range of skin tone values to identify where the hands and face are. With that, OpenCV can generate black and white image with only the outline of the hand. While this step is ideally accurate and quick to implement, the classification of the movement-based letters j and z will continue to be an issue. These letters use the hand positions of other letters but incorporate motion. Past models have circumvented this issue by using the static images at the end of motion to classify them. Thus, true ASL alphabet classification model would require finger tracking to really know what a sign is.

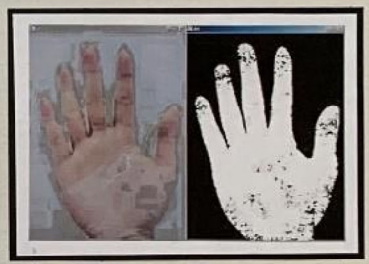

Figure 5: This image shows the process of skin detection with OpenCV.

Conclusions

American Sign Language (ASL) is a visual language and various signs can be classified using static images and machine learning. Hoping to bridge the gap in communication between the deaf or hard of hearing and those who can hear, by using python and Keras' convolutional neural network, I tried to determine the best model structure to get the highest test accuracy while classifying the 24 non-movement based letters of the ASL alphabet. I changed the activation functions, number of hidden layers, number of neurons per layer, max pools, dropouts, epochs, and kernel sizes, to find a sequential model that would give me the highest test accuracy. Though, as the complexity of the neural network increased, it did not always result in an increase of the test accuracy. The model surpassed its proposed accuracy of 95 percent and reached 96.8 percent test accuracy. Although the model initially had difficulties with classifying letters such as p and k that have the same hand position in different orientations, the addition of another hidden layer helped the model find a distinction. Despite these accomplishments, there is still lots of work to be done with the usability of the program and the ability to classify full words.