Catch These Signs Final Project Report

December 2, 2019

Team Members:

Devon Tyson
Philip Cesani
Randall Harrison

GitHub:

https://github.com/Catch-These-Signs/Sign-Language-Project

Contents

1 Introduciton

Sign languages are languages that use the visual-manual modality to convey meaning, expressed through manual articulations in combination with non-manual elements. They are full-fledged natural languages with their own grammar and lexicon. American Sign Language is one of the most commonly used minority languages in the United States. Therefore, there is a demand for technology that can assist the deaf community and those who use sign language with a broader means of communication.

2 Problem Description:

The goal of the project is to better understand and improve communication for the deaf community by using machine learning and computer vision techniques. We will achieve this by using images of signed letters and translating them into text characters that are readable by a computer. We will use a Convolutional Neural Network to take images of signed letters and make predictions about them in order to classify them as their respective letter.

3 Description of Data:

The American Sign Language letter database of hand gestures represent a multi-class problem with 24 classes of letters (excluding J and Z which require motion). Our dataset format is patterned to match closely with the classic MNIST dataset. Each training and test case represent a label (0-25) as a one-to-one map for each alphabetic letter A-Z.

4 Our process:

We reviewed many projects, chose the sign language dataset and analyzed the data within the set. We then determined the attributes and classifications for the project. The attributes include the pixel value for each pixel in each image and the classifications include the corresponding letter of the alphabet. We have collected and prepared the data for use and set up a GitHub organization with a repository for the project. We have downloaded the proper software, reviewed the Python programming language and have begun working on the project in Jupyter Notebook. For the next step, we built the Convolutional Neural Network, Multi-Layer Perceptron, and K-Nearest Neighbor algorithms. We set up all of the training parameters, found the placeholders, biases, and weights to use to train the data for each neural network. Each algorithm was set up with its own loss function and optimizer. We set up the Convolutional and Pooling layers for the CNN algorithm and began training. Once the data was trained for each model, we tested the data and made predictions.

5 Preliminary Plan:

1. Find a project
2. Understand the problem
3. Download the proper software
4. Review and learn the Python language
5. Set up a GitHub Organization with repository
6. Define the project objective
7. Prepare the data
8. Collect the data
9. Select Algorithms
10. Train the model
11. Test the model
12. Make predictions
13. Conclude the results

6 Methods Overview:

We used the sign language data from MNIST with three different algorithms, Convolutional Neural Network, Multilayer Perceptron, and k-Nearest Neighbors in order to compare the results from them.

7 Convolution Neural Network (CNN):

Convolutional Neural Network or CNN is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex

Convolutional Neural Networks have the following layers:
-Convolution
-ReLU Layer
-Pooling
-Fully Connected

CNN compares the images piece by piece. The pieces that it looks for are called features. By finding rough feature matches, in roughly the same position in two images, CNN gets a lot better at seeing similarity that whole-image matching schemes.

7.1 CNN Creation:

CNN will move the feature/filter to every possible position of the image
Step One - Line up the feature and the image
Step Two - Multiply each image pixel by the corresponding feature pixel.
Step Three - Add them up
Step Four - Divide by the total number of pixels in the feature.

Creating a Map to Put the Value of the Filter:
-Now to keep track of where that feature was, we create a map and put the value of the filter at that place.

Sliding the Filter Throughout the image
-Now, using the same feature and move it to another location and perform the filtering again.

Convolution Layer Output
Similarly, we will perform the same convolution with all the other filters

7.2 ReLU Layer

-In this layer we remove every negative value from the filtered images and replace them with zeros.
-This is done to avoid the values from summing up to zero

Rectified Linear Unit (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable.

7.3 Pooling Layer

In this layer we shrink the image stack into a smaller size

Steps:
1. Pick a window size (usually 2 or 3).
2. Pick a stride (usually 2).
3. Walk your window across your filtered images.
4. From each window, take the maximum value.

7.4 Additional Steps

Calculating the Maximum Value in each Window Start with the first filtered image. In the first window the maximum or highest value is 1, so we track that and move the window two strides.

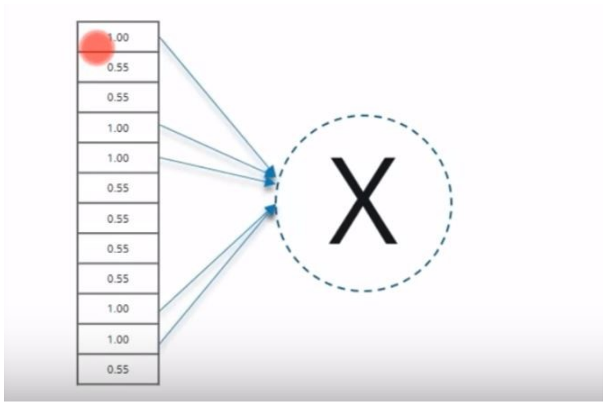
Output After Passing Through Pooling Layer
Reduces the input matrix to a smaller output matrix

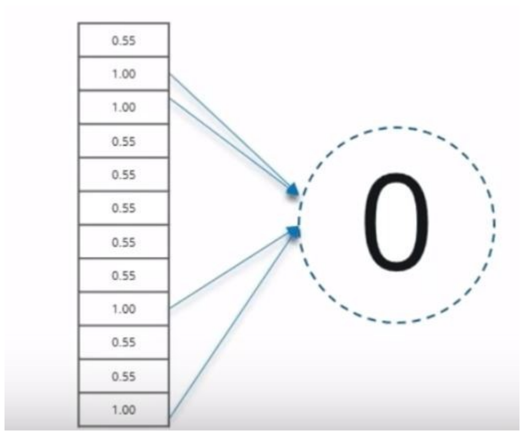
Stacking up the Layers
Adding additional layers of Convolution, ReLU, and Pooling will reduce the output matrix further.

Fully Connected Layer
This is the final layer where the actual classification happens. Here we take our filtered and shrunken images and put them in a single list

7.5 Output

When we feed in, ‘X’ and ‘O’. Then there will be some element in the vector that will be high. Consider the image below, as you can see for ‘X’ there are different elements that are high and similarly, for ‘O’ we have different elements that are high.





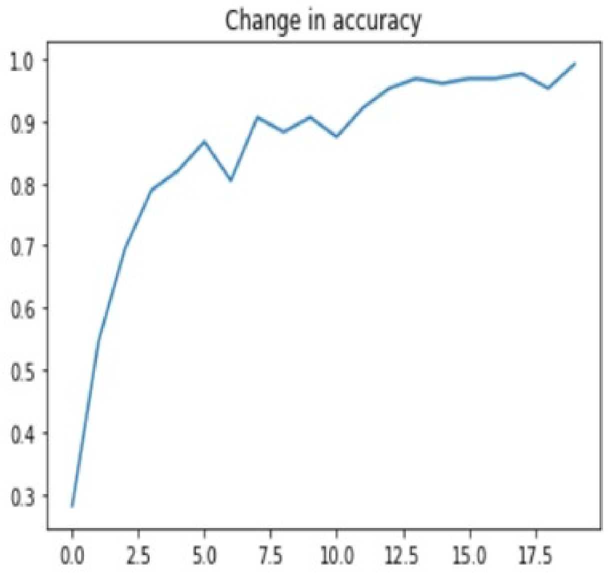
7.6 Prediction

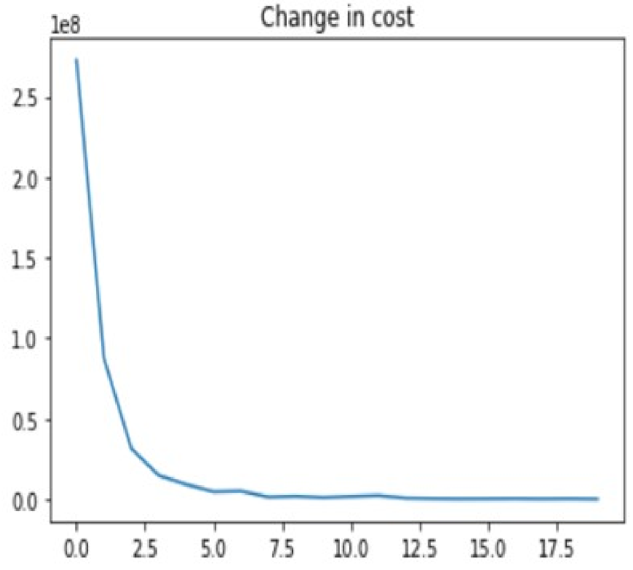
Consider the list of a new input image. A new input image passes through all of the layers

Comparing the Input Vector with X
This new list is compared with the trained list that has been separated into different classes.

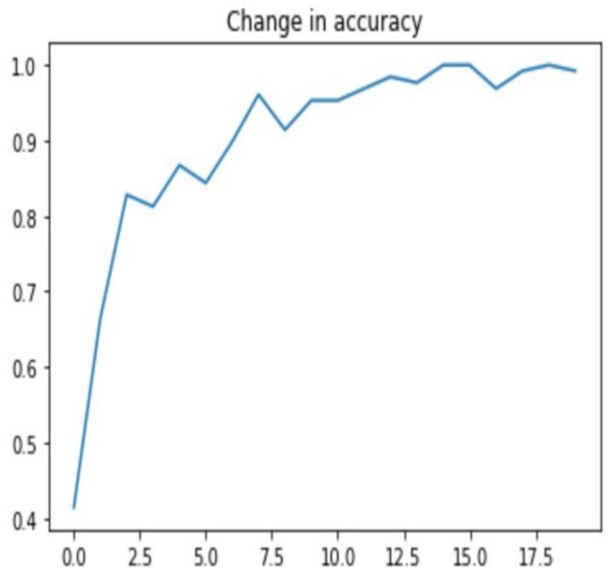
7.7 Results

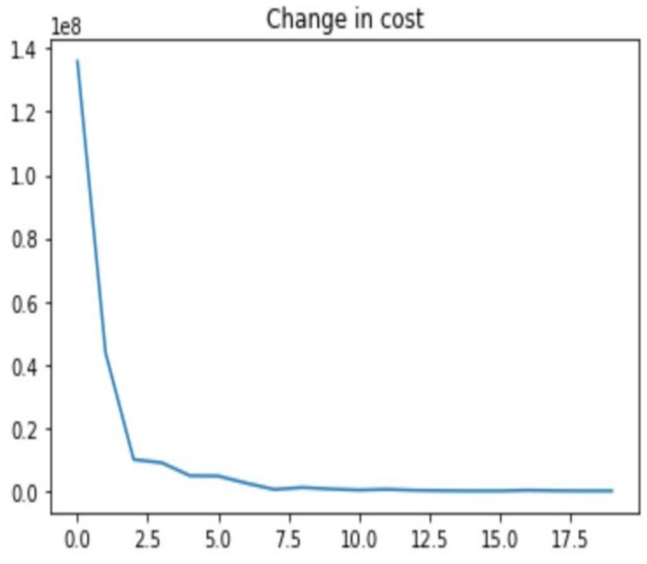
After this is done the input image is identified with some level of accuracy.
Training





Testing





7.8 Implementing the Use-Case

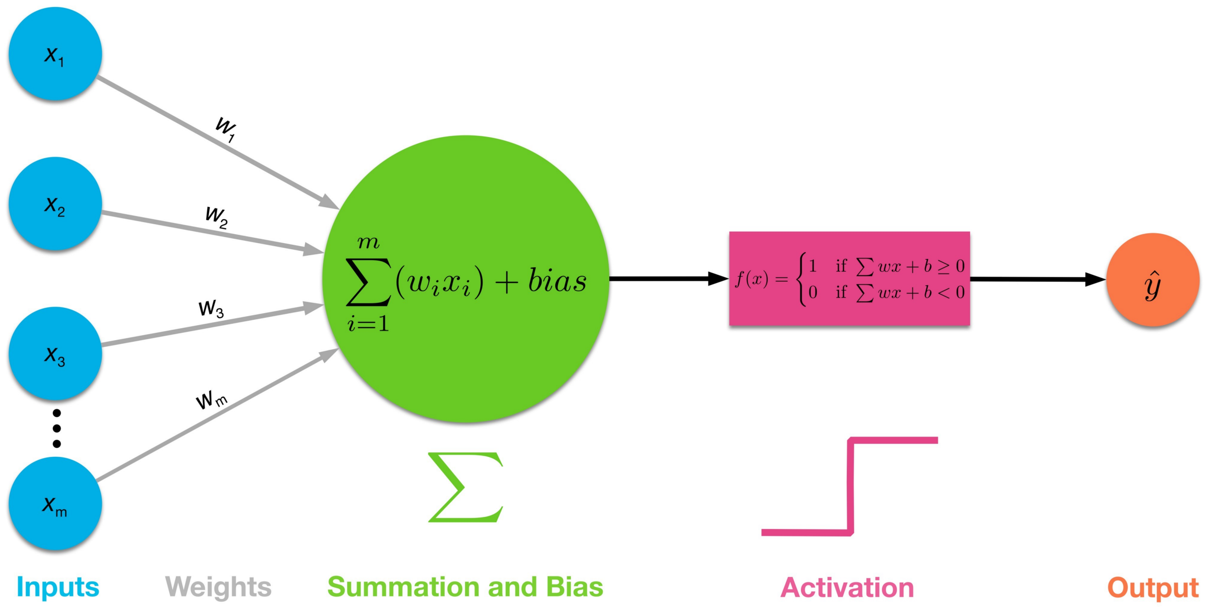
Download the dataset
Function to encode the labels
Resize the image (X) x (X) pixel and read as greyscale

Split the data 50% for training and 50% for testing
Reshape the data appropriately for TensorFlow
Build the Model
Calculate loss function, it is categorical cross entropy
Optimizer with learning rate set to 0.001
Train the Deep Neural Net for (X) epochs
Make predctions

8 Multi-Layer Perceptron (MLP):

Multi-Layer Perceptron is the predecessor to Convolutional Neural Networks. It was once used as image classification due to the use of hidden layers in the neural network. It's good for image classification because it can distinguish data that is not linearly separable. MLP works as a feedforward artificial neural network. It contains many perceptrons, or binary classifiers, that are organized into layers. There are at least 3 layers included – the input layer, a hidden layer and an output layer; however, there may be many hidden layers. Each node is a neuron that uses an activation function in each layer. In our case, we used ReLu activation function. Backpropagation is used for training, which uses the previous layer as an input for the current layer. Learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result. Once all layers have been activated, linear algebra reduces the layers into a two-layer input-output model.

Each layer can be visualized like the example below:

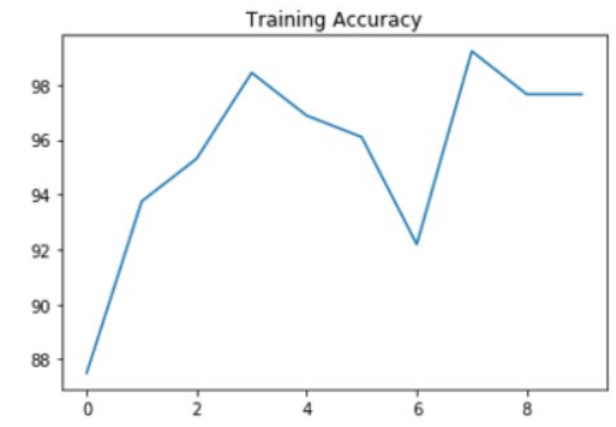

x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output

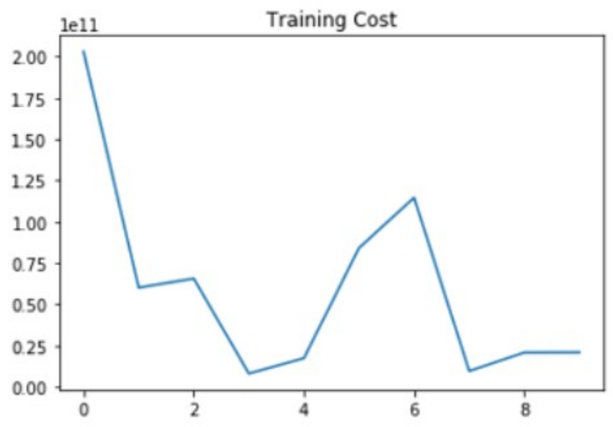
8.1 MLP Settings:

Loss function: Cross Entropy
Optimizer: Adam
Activation Function: ReLu
Layers: 10
Learning Rate: 0.002
Batch size: 128
Epochs: 5000

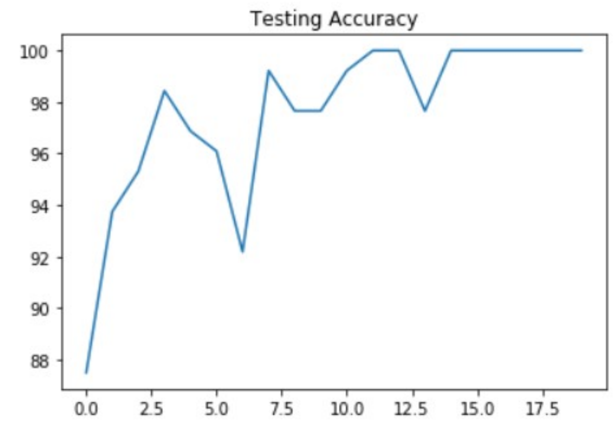
8.2 Results:

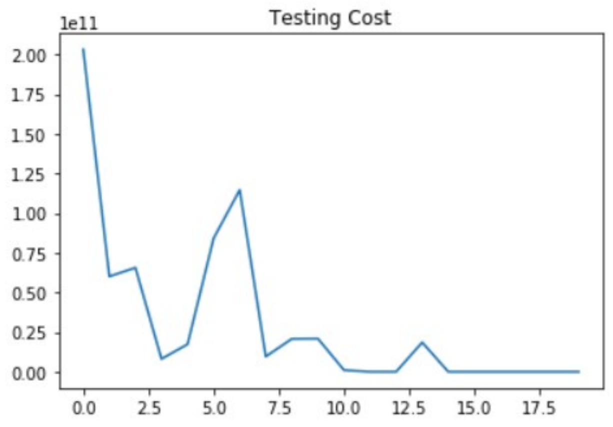
The average training accuracy came out to 96.35%





The average testing accuracy came out to 99.14%

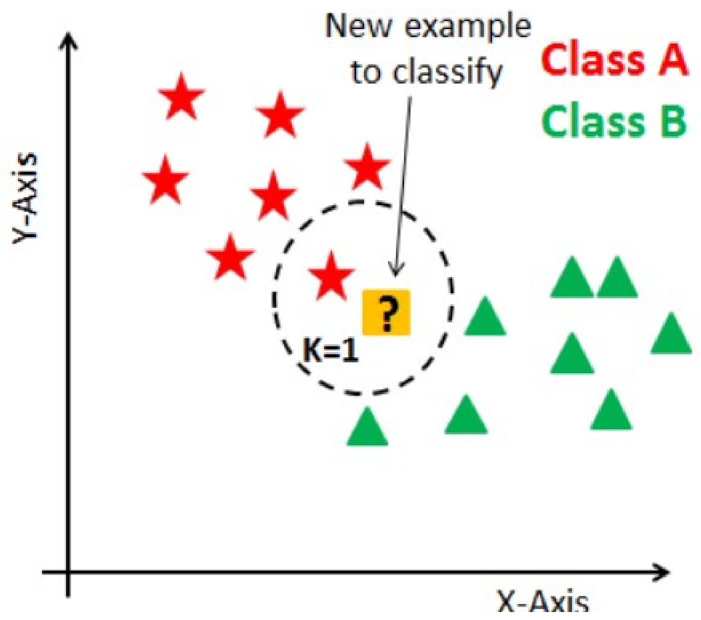




9 K Nearest Neighbors (KNN):

In KNN (K is the # of nearest neighbors) is a non- parametric, lazy algo. It finds the nearest neighbor and uses that neighbor's label to help predict the classification of K. KNN does not make any assumption on the data distribution, so the model structure is determined from the data. KNN also is based on feature similarity which means it uses our training set to classify determine what the classification of a given data point is.

Example:



? = Class A based on model

9.1 Settings For KNN:

We used the sklearn to import out knn classifier
Number of Neighbors = 5
Metric = Minkowski (so we can use the Euclidian distance between point data point to determine classification)
P =2

9.2 Results:

Training Accuracy = 99.22%
Testing Accuracy = 98.83%

10 Conclusion:

Multi-Layer Perceptron has little to no accuracy when it comes to complex images having pixel dependencies throughout. Convolutional Neural Networks are able to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. The role of CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. K-Nearest Neighbor and Convolutional Neural Networks perform competitively, while CNN produces higher accuracy than KNN. For this dataset, a Convolutional Neural Network performs best.

11 Future Work:

A visual recognition algorithm for American Sign Language could provide new benchmarks that challenge modern machine learning methods such as Convolutional Neural Networks and could also help the deaf and hard-of-hearing to better communicate by using computer vision applications. In the future, the results from this project can be used in collaboration with a text-to-speech application with the intention of making communication as easy as possible for the deaf community. It could potentially even make communication faster and easier during conversations between deaf and blind people together.

12 References:

1. Huy V. Vo, Francis Bach, Minsu Cho, Kai Han, Yann LeCun, Patrick Perez, Jean Ponce. Unsupervised Image Matching and Object Discovery as Optimization. In CVPR, 2019.

2. P. Isola, J-Y. Zhou, and A.A. Efros. Image-to-image translation with conditional adversarial networks. In CVPR, pages 1125 - 1134, 2017.

3. J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image to image translation using cycle-consistent adversarial networks. In CVPR, pages 2223–2232, 2017.

4. M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised image-toimage translation networks. In Advances in Neural Information Processing Systems, pages 700–708, 2017

5. Raia Hadsell, Sumit Chopra, Yann LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In CVPR, 2006.