American Sign Language Alphabet Recognition Using Machine Learning

Stefan Ladinski
Ss. Cyril and Methodius University,
Faculty of Computer Science and Engineering
Skopje, North Macedonia
stefan.ladinski.work@gmail.com

Abstract - In this paper, we propose a machine learning model for recognizing all 26 letters in the American Sign Language (ASL) alphabet. The model is trained using a dataset obtained by recording a 30-frame video of hand movements. MediaPipe is used to detect hand positions in each frame and extract their coordinates, resulting in an array of 63 values. These sequences of arrays are then passed down to our Sequential model that uses LSTM as the input layer and Dense as the output layer. We evaluated two models, with Model 1 and Model 2 both achieving similar accuracy. Our study demonstrates that the proposed machine learning model consisting of MediaPipe's hand detector and a neural network can effectively recognize all letters of the ASL alphabet.

Keywords: sign language, alphabet recognition, machine learning.

I. Introduction

Sign language is used all around the world. The language consists of specific movements of the hands and body. Not a lot of people know how to use sign language and that is why the goal of this project is to build a machine learning model that can recognize all the 26 different letters in the American Sign Language (ASL) alphabet. To achieve this goal we will be using a neural network consisting of Long-Short Term Memory (LSTM) and Dense layers.

II. Data

A. Why don't we use a Convolutional Neural Network, with images as an input?

Convolutional Neural Networks or CNNs [1] take one image per input (letter). The problem here is that we can't represent all letters of the ALS alphabet with one image.

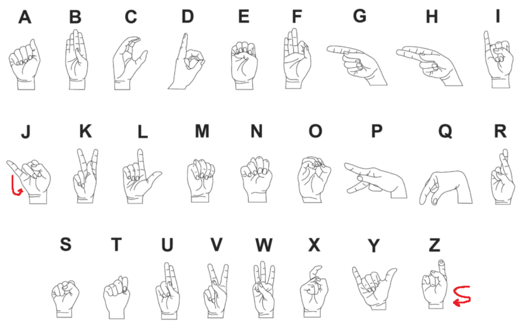


Fig. 1. Sign Alphabet

As shown in Fig. 1 the letters J and Z require movement and thus cannot be represented with a single image.
The solution is to use multiple images to represent each letter. Each letter will be represented by a video consisting of 30 frames. For each letter we will need multiple videos, we will be using 60 videos per letter. Here another problem arises. We have 26 letters and each letter has 60 videos and each video is 30 frames. This means that we will have to keep a total of 1560 videos or 46800 frames/images. Storing so many videos presents a storage problem.

B. Using Arrays Instead of Images

Instead of storing every video individually we decided to extract the positions of the left and right hand in every frame and store that. To detect the hands [2] and their landmarks we used the MediaPipe framework. MediaPipe detects 21 different landmarks on each hand. Each landmark has a relative X, Y and Z coordinate.

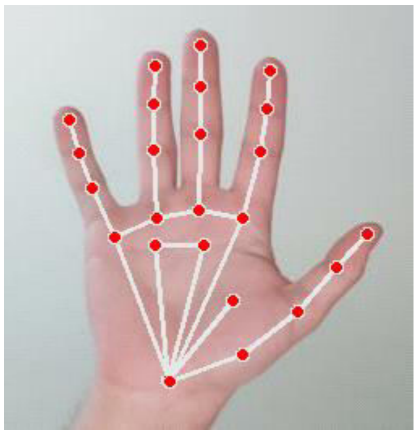
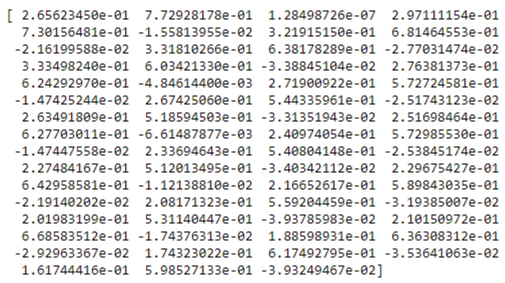


Fig. 2. Detection of hand coordinates

Using MediaPipe we can detect and extract the X, Y and Z coordinates of all 21 landmarks from each hand from a single frame. After the landmarks are detected and the coordinate information is extracted we will have an array of 126 values. The array contains 126 values because we have 2 hands, each hand has 21 landmarks and each landmark has 3 coordinates. Using this method we have greatly reduced the storage requirements while also retaining the information about the position of the hands in each frame.



```
[ 2.65623450e-01  7.72928178e-01  1.28498726e-07  2.97111154e-01
  7.30156481e-01 -1.55813955e-02  3.21915150e-01  6.81464553e-01
 -2.16199588e-02  3.31810266e-01  6.38178289e-01 -2.77031474e-02
  3.33498240e-01  6.03421330e-01 -3.38845104e-02  2.76381373e-01
  6.24292970e-01 -4.84614400e-03  2.71900922e-01  5.72724581e-01
 -1.47425244e-02  2.67425060e-01  5.44335961e-01 -2.51743123e-02
  2.63491809e-01  5.18594503e-01 -3.31351943e-02  2.51698464e-01
  6.27703011e-01 -6.61487877e-03  2.40974054e-01  5.72985530e-01
 -1.47447558e-02  2.33694643e-01  5.40804148e-01 -2.53845174e-02
  2.27484167e-01  5.12013495e-01 -3.40342112e-02  2.29675427e-01
  6.42958581e-01 -1.12138810e-02  2.16652617e-01  5.89843035e-01
 -2.19140202e-02  2.08171323e-01  5.59204459e-01 -3.19385007e-02
  2.01983199e-01  5.31140447e-01 -3.93785983e-02  2.10150972e-01
  6.68583512e-01 -1.74376313e-02  1.88598931e-01  6.36308312e-01
 -2.92963367e-02  1.74323022e-01  6.17492795e-01 -3.53641063e-02
  1.61744416e-01  5.98527133e-01 -3.93249467e-02]
```

Fig. 3. Coordinates of the landmark for one hand

III. Datasets

A. Creating Dataset 1

The problem required a dataset of videos where each video will represent one letter. We were unable to find such a dataset and that is why we decided to create our own dataset. Prior to collecting the data for the dataset we had to learn the letters of the ASL alphabet.

The data is collected with a loop that runs 30 times for each letter. Every time the loop runs it records a video that consists of 30 frames. As the frames are being recorded we use MediaPipe to detect and extract the landmark coordinates of the hands for every frame. The process is repeated 30 times for each letter, this way for each letter there are 30 sequences 30 of arrays.

B. Creating Datasets II & III

MediaPipe detects the hand coordinates relative to the entire video feed. This means that the hand can be anywhere in the frame, therefore resulting in more unnecessary data variation. The method used to create Datasets 2 & 3 is similar to the method used to create Dataset 1 only with one difference. After MediaPipe extracts the hand coordinates, instead of saving them as they are we scale and normalize them. The wrist landmark's (landmark 0) coordinates are set 0 and all other landmark coordinates are scaled accordingly. This results in dataset with much less data variation.

Dataset 2 uses the X and Y coordinates of each hand landmark, while Dataset 3 uses all three coordinates for each hand landmark.

C. Data Split

Once we have a sufficient dataset the next step is to use the data to train the machine learning model. First the dataset has to be split into separate sets: one for training, one for validation and the other for testing. In this project all datasets use the following split:

- Training set contains 70% of the original dataset.
- Validation set contains 15% of the original dataset.
- Testing set contains 15% of the original dataset.

IV. Applied models

The model that was used in this project is a Sequential model [3] from the Tensorflow library. The sequential model is a neural network where we can add layers. Each layer's input is the output of the previous layer. The first layer in this model is the input layer. The last layer is the output layer. The output layer returns a probability for each class in the data that tells us the likelihood that the class corresponds to the input. All models used in this project use LSTM as an input layer and Dense as an output layer.

Layers:

- LSTM (Long short-term memory) layers [4] are layers that can learn from sequential data.
- Dense layers [5] are layers where all the input nodes of the layer are connected to all the output nodes of the previous layer.
- Dropout layer [6] is a layer used to combat overfitting. The layer choses a number of output nodes from the previous layer and it disregards their output.

Model 1 is a sequential model consisting of:

- LSTM layer with an input of (30, 42 or 63) and output dimensionality of 64.
- LSTM layer with an output dimensionality of 128.
- LSTM layer with an output dimensionality of 64.
- Dense layer with an output dimensionality of 64.
- Dense layer with an output dimensionality of 32.
- Dense layer with an output dimensionality of 26 that corresponds to the 26 letters in the dataset.

Model 2 is a sequential model consisting of:

- LSTM layer with an input of (30, 42 or 63) and output dimensionality of 128.
- LSTM layer with an output dimensionality of 128.
- Dropout layer that drops 20% of the neurons for the previous layer.
- Dense layer with an output dimensionality of 26 that corresponds to the 26 letters in the dataset.

A. Results

Long short-term memory is a type of neural network architecture used to classify sequential data. Each letter is represented by a sequence of arrays so this type of architecture is suited. Using the dataset that was collected we tested many different configurations of the neural network, these were the best results:

Model 1 is a neural network consisting of three LSTM layers and three Dense layers. The model was tested with all three Datasets.

- Dataset 1:
  - Accuracy score = 0.70
  - F1 score = 0.57
- Dataset 2:
  - Accuracy score = 0.92
  - F1 score = 0.90
- Dataset 2:
  - Accuracy score = 0.89
  - F1 score = 0.88

Model 3 uses less layers than the previous model. Model 3 has two LSTM layers, one Dropout layer (used to combat overfitting) and one Dense layer.

- Dataset 1:
  - Accuracy score = 0.64
  - F1 score = 0.63
- Dataset 2:
  - Accuracy score = 0.98

- F1 score = 0.95
- Dataset 2:
  - Accuracy score = 0.95
  - F1 score = 0.96

B. Testing

The performance of all the models was measured by testing their F1 score and Accuracy score using the testing dataset. It is useful to visualize this information using a matrix. The diagonal of the matrix shows us how many times each letter has been guessed correctly (darker color means more guesses). We can also see how many times each specific letter has been misclassified as another letter.

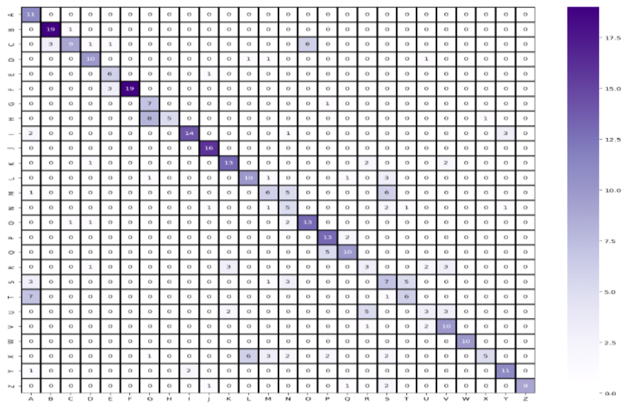


Fig. 4. Results from the testing dataset

V. Conclusion

Storing individual videos to use as data takes a lot more space than just storing the coordinates of the hands in each frame of the video. The dataset that we are using in this project consists of 1560 videos. That means that for every letter in the alphabet we have 60 sequences of arrays representing it. The dataset was created specifically for use in this project. All the models that were tested showed the same trend: Fairly high results while training, but lower results while testing. The fairly high results achieved while training indicate that the models are able to learn from the data. The lower results while testing might indicate issues with the models, but I think the more likely culprit are errors in the dataset. Since this dataset was created manually with amateur equipment it is prone to errors.

References

[1] "tf.keras.layers.Conv2D," TensorFlow Documentation. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

[2] "MediaPipe," Google Developers. [Online]. Available: https://developers.google.com/mediapipe

[3] "tf.keras.Sequential," TensorFlow Documentation. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

[4] "tf.keras.layers.LSTM," TensorFlow Documentation. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

[5] "tf.keras.layers.Dense," TensorFlow Documentation. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

[6] "tf.keras.layers.Dropout," TensorFlow Documentation. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

[7] "sklearn.metrics.f1_score," scikit-learn Documentation. [Online]. Available: https://scikitlearn. org/stable/modules/generated/sklearn.metrics.f1_score.htm

[8] "CNN using Keras - 100% Accuracy," Kaggle. [Online]. Available: https://www.kaggle.com/code/madz2000/cnn-usingkeras-100-accuracy

[9] "Deep Learning using Sign Language," Kaggle. [Online]. Available: https://www.kaggle.com/code/ranjeetjain3/deeplearning-using-sign-langugage