An Automated Recognition System of Sign Language for Static Signs using Deep Learning

Problem Statement

Speech impaired people use hand-based gestures to communicate. Unfortunately, the vast majority of the people are not aware of the semantics of these gestures. The alternative of written communication is very difficult because the Deaf community is generally less skilled in writing a spoken language. Furthermore, this type of communication is impersonal and slow in face-to-face conversations. In an attempt to bridge the same, we propose a real-time hand gesture recognition system based on the data captured by a Web camera. The purpose of this work is to contribute to the field of automatic sign language recognition. We focus on the recognition of the signs or gestures. Our model was successful in surpassing state of the art testing. The highest generalized testing accuracy that we are able to achieve is 98.56 per cent on validation data and 99.91 per cent on test data. In future, we aspire to add Natural Language Processing so the model can make words and sentences out of the letters it recognizes. This can then be used practically by people.

Background

Sign language is a visual language that uses gestures as a means of communication. There are many sign languages all around the world including American Sign Language (ASL), Indian Sign Language (ISL), British Sign Language (BSL), Australian Sign Language (Auslan). All these languages are gesture based. So, Understanding human gestures can be presented or assumed as a pattern recognition problem. If these gesture patterns are detected and distinguished by computers then the desired language can be reconstructed. Our research on sign language translation contributes a variety of statistical computer vision used to signify alphabets and numbers and recognize the sign language more efficiently. Here, we have used machine learning and Convolution Neural Network (CNN) based technique as the recognizer of the system, we have used American Sign Language (ASL) as sign language to detect the gestures we attempt.

ASL is most widely used sign language among world in more than 20 nations. In some nations it is used as the primary based communication. ASL is a striking form of communication favorable to large portion of impairment people. The global impacts of these sign recognition are quite amazing and eye-opening. ASL provides a set of 26 gesture signs names as an American Manual Alphabet that can be cast-off to spell out many of English words available. The 26 American Manual Alphabets can be made by 19 various hand shapes of ASL. ‘K’ and ‘P’ alphabets use the identical hand shape with diverse orientations. ASL also offers numeric gestures to sign the numbers from ‘0’ to ‘9’. Built-in ASL equivalent signs for technical terms and accurate nouns are not comprised in ASL. Our proposed system aims at recognizing static ASL gestures and converts them into corresponding letters. As it is a vision based approach web camera is used for obtaining the data from the signer and it can also be used offline. The angle between the hands is strict when it comes to ASL. So, the purpose of this system is to serve as most efficient way of communication for impairs people.

Methodology

Step 1: Setting an environment

The environment required for the implementation has to be set up before building the model. This includes importing and installing all the necessary packages and libraries such as Keras, Matplotlib, Numpy, Pandas, Pillow, Seaborn, sklearn and Tensorflow.

Step 2: Analyzing data and splitting the train, test and validate data

Dataset is split in Train, Validate and Test sets at the ratio of 70:20:10. With a total of 34,627 images 25,055 are Training samples. 2,400 are Training samples and 7172 are Validation samples. The high validation data helps better the learning rate. The data is in the form of CSV containing 785 columns, one for the label and the other 784 for the pixels in 28×28 images.

Step 3: Preprocessing data

To process the data the training samples are used. The 784 columns must be converted to 2 dimensional 28×28 columns. All the pixels are then normalized between 0 and 255. Since more the input to the model when training, better the model. Later Data Augmentation is performed on the training data. Minor augmentations like tilting an image by 10 degrees or shifting an image by some pixels are performed, and this increases our training data almost two-fold. The preprocessed data is now ready for training the model. Data Augmentation is performed on train data.

Step 4: Modeling

The model consists of 3 convolutional layers of 3×3×1 each followed by a Max Pooling layer (see Figure 1) The three convolutional layers have 75, 50 and 25 activation layers in order. Dense and dropout layers are added to give the neural network more inputs for further layers. Dropout has a probability of 0.2. Convolutional layers use Rectified Linear Unit, ReLu, as its activation function. The end of the neural network uses a Softmax function to get class predictions from the input data. We have trained our model compiled with a batch size of 128 for 60 epochs. The model is then saved for further use. The Keras model, which is now saved locally, can be loaded anytime and used anywhere.

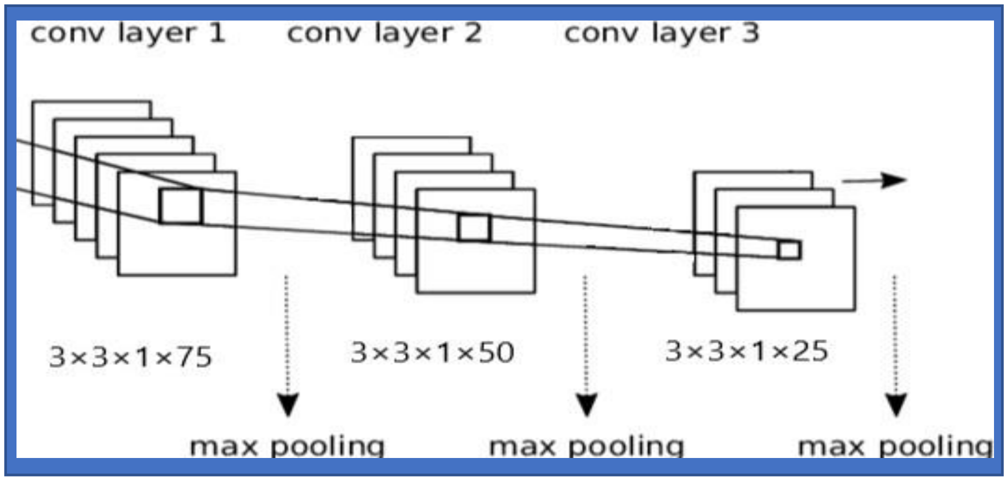

Figure 1. Shows the Architecture of the model

Experimental Design

Dataset

The Dataset used to build the model is MNIST Dataset from Kaggle [9] which has about 27455 pixel data for training and over 7172 pixel data for validation. Dataset is split in Train, Validate and Test sets at the ratio of 70:20:10. The Dataset has been modified as mentioned in the Proposed Methodology. The training set is further put to Data Augmentation to almost double or even triples the inputs. Remember to not flip or rotate images as they might lead to misclassification.

Model Evaluation

Using web cam, we uploaded a static sign gesture to evaluate the model's accuracy. It gave us an accuracy of 99.1% on test data and an accuracy of 98.56% on validation data.

Software and Hardware Requirements

Python based Computer Vision and Deep Learning libraries will be exploited for the development and experimentation of the project. Tools such as Anaconda Python, and libraries such as Keras, Matplotlib, Numpy, Pandas, Pillow, Seaborn, sklearn and Tensorflow will be utilized for this process. Training will be conducted on NVIDIA GPUs.