International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 03 | March - 2022 ISSN: 2582-3930

Sign Language Recognition

Pariksheet Shende, Rajat Bais, Priyanka Karamkar, Rushika Bhave, Jayesh Mankavade

PRIYADARSHINI COLLEGE OF ENGINEERING, NAGPUR

Abstract

This paper focuses on experimenting with different segmentation approaches and unsupervised learning algorithms to create an accurate sign language recognition model. To more easily approach the problem and obtain reasonable results, we experimented with just up to 10 different classes/letters in our self-made dataset instead of all 26 possible letters. We collected 12000 RGB images and their corresponding depth data using a Microsoft Kinect. Up to half of the data was fed into the autoencoder to extract features while the other half was used for testing. We achieved a classification accuracy of 98% on a randomly selected set of test data using our trained model. In addition to the work we did on static images, we also created a live demo version of the project which can be run at a little less than 2 seconds per frame to classify signed hand gestures from any person.

1. Introduction

The problem we are investigating is sign language recognition through unsupervised feature learning. Being able to recognize sign language is an interesting computer vision problem while simultaneously being extremely useful for deaf people to interact with people who don't know how to understand American Sign Language (ASL). Our approach was to first create a data set showing the different hand gestures of the ASL alphabet that we wished to classify. The next step was to segment out only the hand region from each image and then use this data for unsupervised feature learning using an autoencoder, followed by training a softmax classifier for making a decision about which letter is being displayed. Many different segmentation approaches were tried until we discovered that the skin color and depth segmentation techniques worked most consistently.

2. Methodology

2.1. Problem Statement

2.1.1 Background: The CVPR gesture workshop from 2011 provides a great information on modern gesture recognition models as well as how to incorporate different learning algorithms. There is some past work1 related to our project that we initially looked at such as segmentation-robust modeling for sign language recognition [3] and sign language and human activity recognition [1], but we ended up using mostly our own approach to sign language recognition. Inspiration for our learning model was drawn from the MNIST handwritten digits recognition problem2 which also used a similar unsupervised feature learning and classification approach [2]. We also investigated the use of convolutional neural networks for feature learning based on the visual document analysis paper by the Microsoft Research group [4]. We felt that a simpler approach, however, would still yield competitive accuracy results.

2.1.2 Dataset: The data used for training and testing came from our own self-made dataset. 1200 samples of each of the 10 signed letters (a, b, c, d, e, f, g, h, i, l) were collected using the Kinect including the corresponding depth data. Each sample consists of a person signing the corresponding letter while facing

directly at the Kinect camera. This dataset consists of 6000 images used for training and another 6000 images used for testing.

3. Sources of Data

3.1 Data Collection

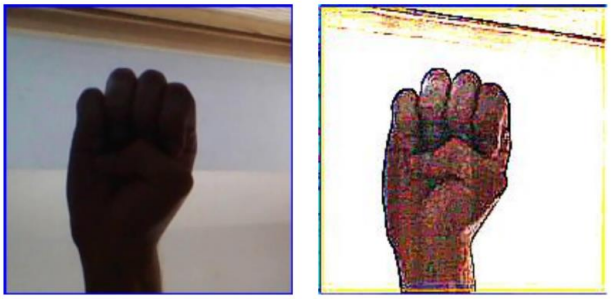
The primary source of data for this project was the compiled dataset of American Sign Language (ASL) called the ASL Alphabet from Kaggle user Akash [3]. The dataset is comprised of 87,000 images which are 200x200 pixels. There are 29 total classes, each with 3000 images, 26 for the letters A-Z and 3 for space, delete and nothing. taken from his laptop's webcam. These photos were then cropped, rescaled, and labelled for use.

(a) ASL letter A (b) ASL letter E (c) ASL letter H (d) ASL letter Y

Image Enhancement: A combination of brightness, contrast, sharpness, and color enhancement was used on the images. For example, the contrast and brightness were changed such that fingers could be distinguished when the image was very dark.

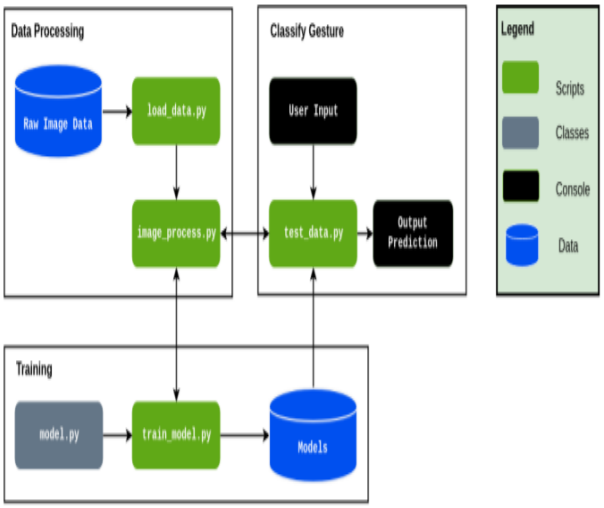
Edge Enhancement: Edge enhancement is an image filtering techniques that makes edges more defined. This is achieved by the increase of contrast in a local region of the image that is detected as an edge. This has the effect of making the border of the hand and fingers, its boundaries.

Image Whitening: ZCA, or image whitening, is a technique that uses the singular value decomposition of a matrix. This algorithm decorrelates the data, and removes the redundant, or obvious, information out of the data. This allows for the neural network to look for more complex and sophisticated relationships, and to uncover the underlying structure of the patterns it is being trained on. The covariance matrix of the image is set to identity, and the mean to zero.



(a) (b) Preprocessed Image

4. Description of Overall Software Structure



As shown in Figure 1, the project will be structured into 3 distinct functional blocks, Data Processing, Training, and Classify Gesture. The block diagram is simplified in detail to abstract some of the minutiae:

- **Data Processing:** The load data.py script contains functions to load the Raw Image Data and save the image data as numpy arrays into file storage. The process data.py script will load the image data from data.npy and preprocess the image by resizing/rescaling the image, and applying filters and ZCA whitening to enhance features. During training the processed image data was split into training, validation, and testing data and written to storage. Training also involves a load dataset.py script that loads the relevant data split into a Dataset class. For use of the trained model in classifying gestures, an

individual image is loaded and processed from the filesystem.

• **Training:** The training loop for the model is contained in train model.py. The model is trained with hyper parameters obtained from a config file that lists the learning rate, batch size, image filtering, and number of epochs. The configuration used to train the model is saved along with the model architecture for future evaluation and tweaking for improved results. Within the training loop, the training and validation datasets are loaded as Data loaders and the model is trained using Adam Optimizer with Cross Entropy Loss. The model is evaluated every epoch on the validation set and the model with best validation accuracy is saved to storage for further evaluation and use. Upon finishing training, the training and validation error and loss is saved to the disk, along with a plot of error and loss over training.

• **Classify Gesture:** After a model has been trained, it can be used to classify a new ASL gesture that is available as a file on the file system. The user inputs the file path of the gesture image and the test data.py script will pass the file path to process data.py to load and preprocess the file the same way as the model has been trained.

5 Machine Learning Model

5.1 Overall Structure The model used in this classification task is a fairly basic implementation of a Convolutional Neural Network (CNN). As the project requires classification of images, a CNN is the go-to architecture. The basis for our model design came from Using Deep Convolutional Networks for Gesture Recognition in AmericanSign Language paper that accomplished a similar ASL Gesture Classification task [4]. This model consisted of convolutional blocks containing two 2D Convolutional Layers with ReLU activation, followed by Max Pooling and Dropout layers. These convolutional blocks are repeated 3 times and followed by Fully Connected layers that eventually classify into the required categories. The kernel sizes are maintained at 3 X 3 throughout the model.

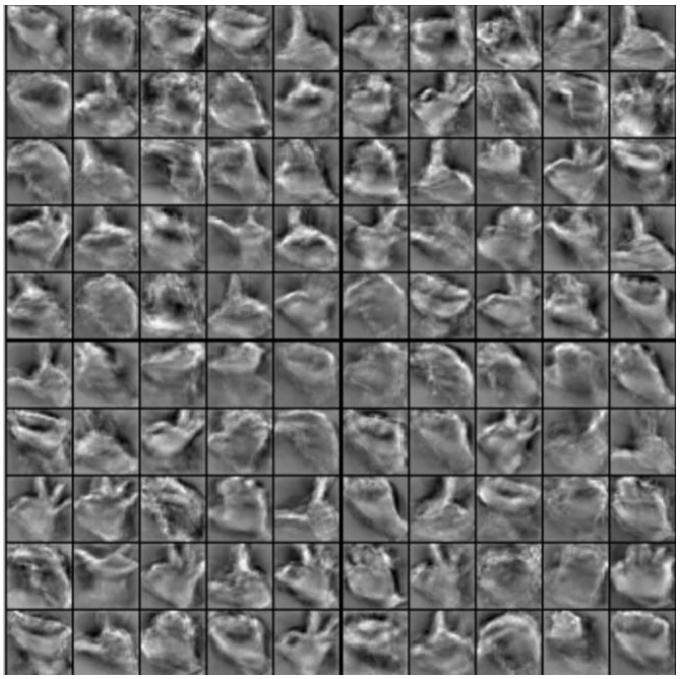


Figure: Visualization of sparse autoencoder features

The next step is to classify the 10 different letters based on the features learnt by the autoencoder training. The output of the hidden layer of the autoencoder is fed into a softmax classifier to now classify the data into 10 categories. The softmax classifier again learns using the L-BFGS optimization function. This algorithm converges after about 40 iterations. We tested the system accuracy by using the remaining 600 images per letter (for a total of 6000 images) as our test set.

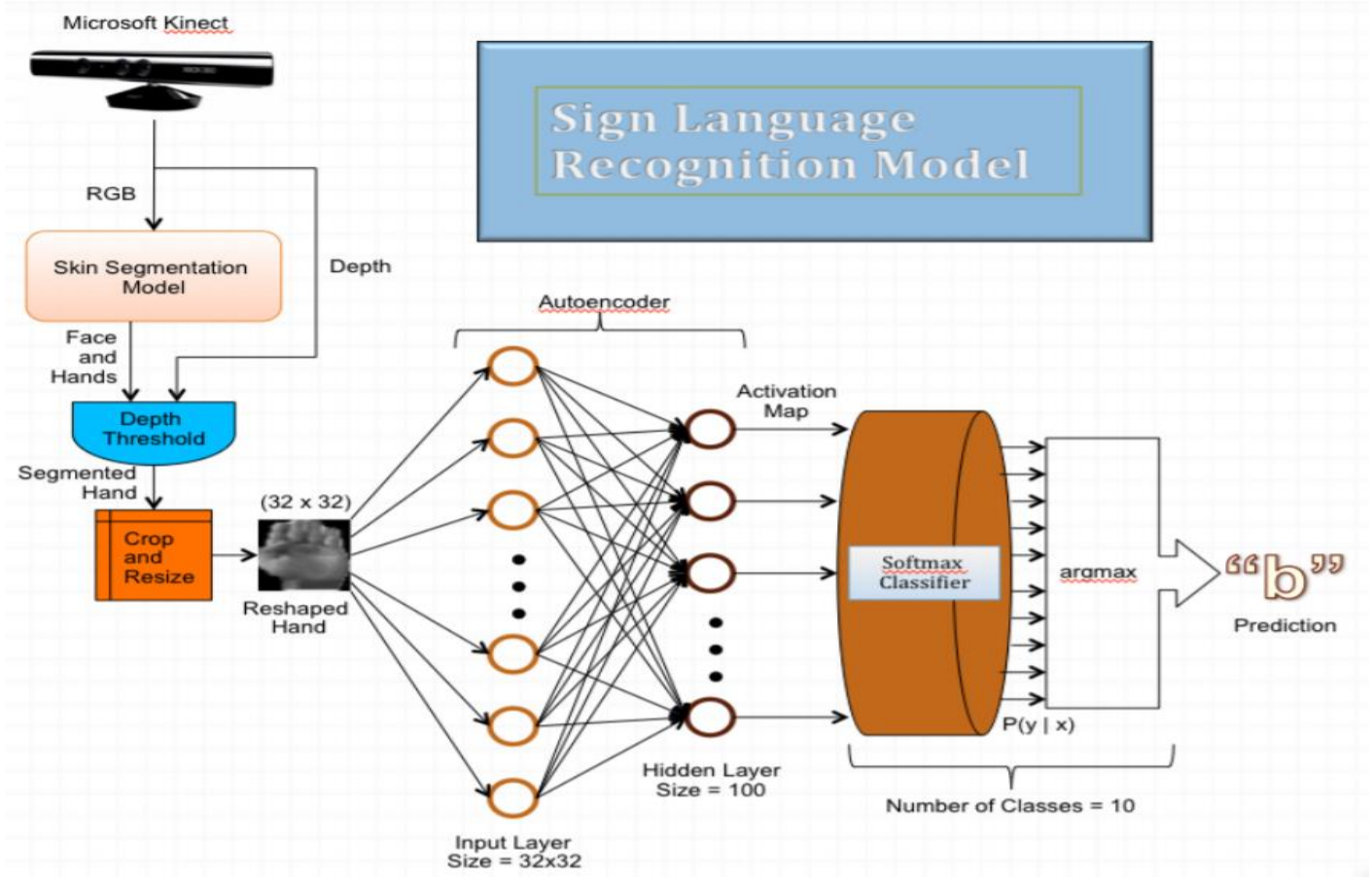


Figure : Block diagram summarizing our approach for sign language recognition

As a finishing step to our project, we have successfully created a real time implementation of our entire system, so that hand gestures made in front of the Kinect connected to our computer directly displayed the image captured by the kinect, the segmented hand gesture and the output of our classifier, which is one of the ten letters in our dataset. The evaluation process takes less than 2 seconds per frame. The following figures show screenshots of our real-time implementation and the results obtained. In each screenshot, the original image is shown with the result of the segmentation and the predicted result to the right of it.

Figure : Example screenshots of real-time live demo sign language recognition system interface

6. Conclusion and Future Work In this project, we have implemented an automatic sign language gesture recognition system in real-time, using tools learnt in computer vision and machine learning. We learned about how sometimes basic approaches work better than complicated approaches. Despite trying to use a smart segmentation algorithm, the relatively basic skin segmentation model turned out to extract the best skin masks. We also realized the time constraints and difficulties of creating a dataset from scratch. Looking back, it would have been nice to have had a dataset already to work off of. Some letters were harder to classify in our live demo such as" a" vs. "i" since they only differ by a very small edge (the "i" has the pinky pointing up). Although our classification system works quite well as has been demonstrated through tables and images, there's still a lot of scope for possible future work

References

[1] D. Metaxas. Sign language and human activity reconition, June 2011. CVPR Workshop on Gesture Recognition.

[2] M. Ranzato. Efcient learning of sparse representations with an energy-based model, 2006. Courant Institute of Mathematical Sciences.

[3] S. Sarkar. Segmentation-robust representations, matching, and modeling for sign language recognition, June 2011. CVPR Workshop on Gesture Recognition, Co-authors: Barbara Loeding, Ruiduo Yang, Sunita Nayak, Ayush Parashar.

[5] X. Teng. A hand gesture recognition system based on local linear embedding, April 2005. Journal of Visual Languages and Computing