RESEARCH PAPER

SIGN LANGUAGE RECOGNITION

Name : Jayakrishnaa kaliki

STU ID: STU62fa89f3e31d71660586483

Abstract

Sign language has indelibly become the ultimate panacea and is a very powerful tool for individuals with hearing and speech disability to communicate their feelings and opinions to the world. It makes the integration process between them and others smooth and less complex. However, the invention of sign language alone, is not enough .There are many strings attached to this boon .The sign gestures often get mixed and confused for someone who has never learnt it or knows it in a different language. However, this communication gap which has existed for years can now be narrowed with the introduction of various techniques to automate the detection of sign gestures.

Sign language is used by deaf and hard hearing people to exchange information between their own community and with other people. Computer recognition of sign language deals from sign gesture acquisition and continues till text/speech generation. Sign gestures can be classified as static and dynamic. However static gesture recognition is simpler than dynamic gesture recognition but both recognition systems are important to the human community. The sign language recognition steps are described in this survey. The data acquisition, data preprocessing and transformation, feature extraction, classification and results obtained are examined. Some future directions for research in this area also suggested.

Introduction

Overview

Artificial Intelligence is a tool that has the potential to make our lives easier and, in some cases, remove obstacles. For example, Artificial Intelligence could be used to take video feeds of people using sign language and translating the signs into words. This could help remove communication obstacles with sign language users with their coworkers and personal relationships.

Such an application would consist of computer vision and natural language processing, but the first step would be to accurately detect and classify sign language signs using computer vision.

In this notebook we will be applying a convolutional neural network to classify a dataset of American Sign Language alphabet images.

About the Data

The dataset consist of 27,455 training images and 7,172 test images. Each image is grayscaled and has a 28 x 28 pixel structure. Each images includes the label of which letter in the alphabet the image represents. There are 24 letters in this dataset, 'j' and 'z' are not included because they both involve motion.

Purpose of this Project

Our task will be to create a CNN model that can accuracy predict the alphabet letter for each image. I am setting out to achieve an accuracy of at least 98%.

Block Diagram

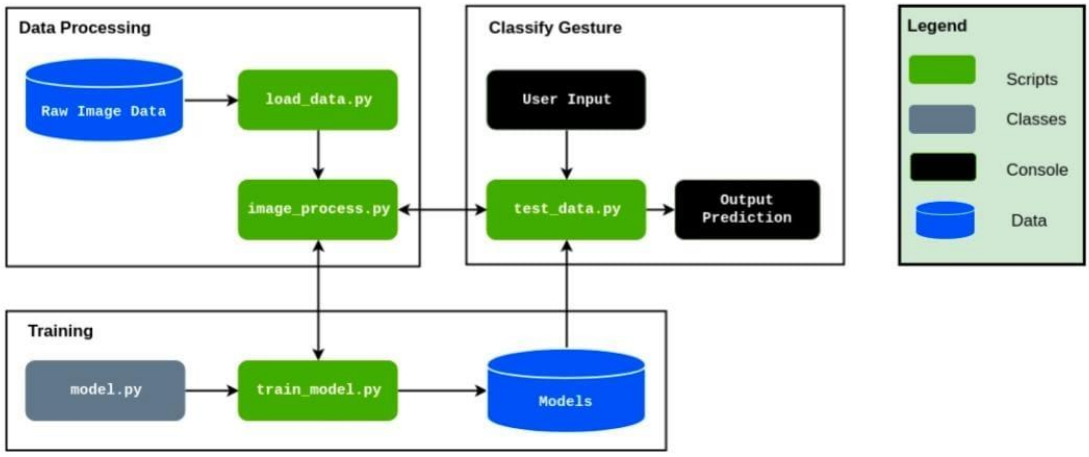


Figure : Block Diagram of Software

As shown in Figure , the project will be structured into 3 distinct functional blocks, Data Processing, Training ,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 hyperparameters 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.

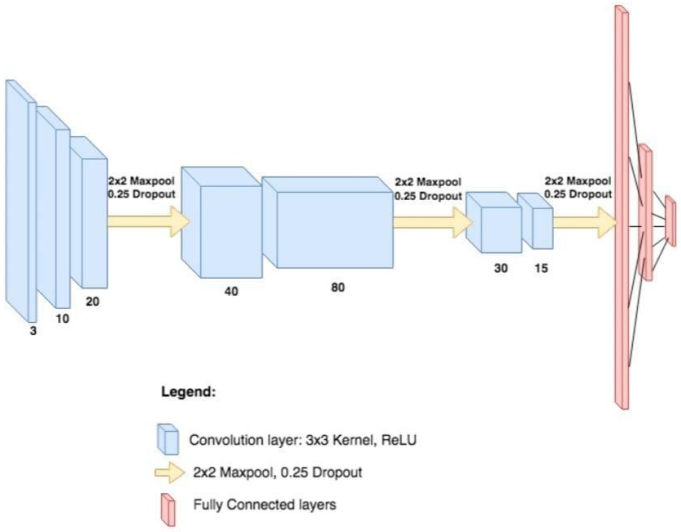
• **Classify Gesture:** After a model has been trained, it can be used to classify a new ASL gesture that is available as a file on the filesystem. The user inputs the file path of the gesture image and the test data. script will pass the file path to process data.py to load and preprocess the file the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

Machine Learning Model and Technologies

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 Sign Language paper that accomplished a similar Sign Language Gesture Classification task . 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.

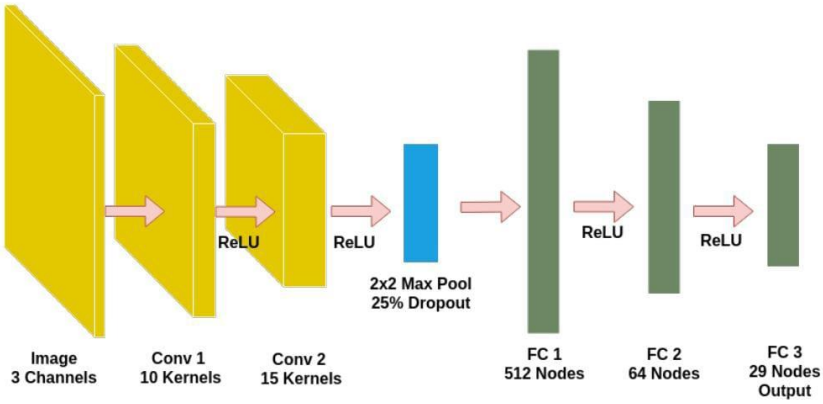


Model Architecture as implemented in Using Deep Convolutional Networks for Gesture Recognition in Sign Language

Model Performance

- **Training And Validation**

Our models were trained using Adam optimizer and Cross Entropy Loss. Adam optimizer is known for converging quickly in comparison with Stochastic Gradient Descent (SGD), even while using momentum. However, initially Adam would not decrease our loss thus we abandoned it to use SGD. Debugging Adam optimizer after our final



Samples/Class	Avg. Validation Accuracy
25	27.3%
50	34.8%
100	46.1%
200	58.2%

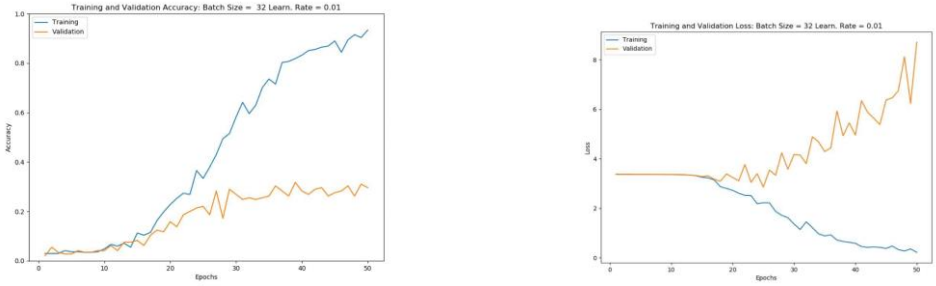
Table 1: Effect of Increasing Dataset Size on Validation Accuracy

Training our initial models on less data led to the models quickly overfitting as shown in figure This is likely due to the small amount of samples to train on leading to bad generalization and learning of the sample space. Increasing the size of our dataset to 200 samples/class led to better model results, with peak validation accuracy

of 60.3% in epoch 17. However, taking a look at our loss function, we see that the validation loss is increasing, indicating overfitting of the model. After we applied filtering, enhancement, and ZCA whitening to our dataset, the model performance increased drastically as shown in Figure. The peak validation accuracy achieved is 77.25% in epoch 24. As shown by the plot of loss, the validation loss is still decreasing, albeit at a slower rate than the training loss, indicating that the model is not drastically overfitting. This shows that preprocessing our images by applying filters and ZCA whitening helps to enhance relevant features for the model to learn.

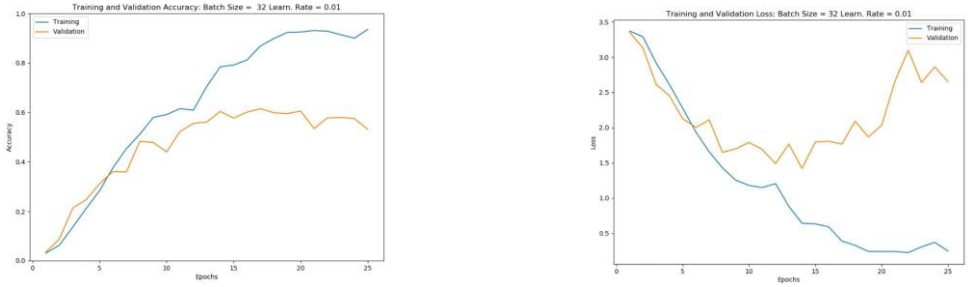
- **Testing**

To determine whether our preprocessing of images actually results in a more robust model, we verified on a test set comprised of images from the original dataset, and our own collected image data. The performance of the models on the test set is shown in Table. We see that the model trained on preprocessed images performs much better than the model trained on the original images, likely due to the former's lack of overfitting. Taking a look at the confusion matrix for the Filtered Model on the Kaggle test set in Figure, we can see that



(a) Training and Validation Accuracy (b) Training and Validation Loss

Figure : Training and Validation Performance of Initial Model Trained on Small Dataset of 50 Samples/Class Demonstrating Overfitting



(a) Training and Validation Accuracy (b) Training and Validation Loss

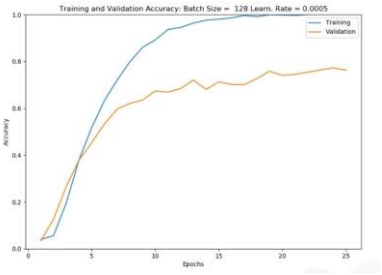
Figure : Training and Validation Performance of Model Trained on Original Images

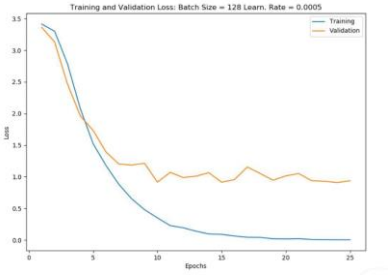
the model is fairly good at predicting the correct letter.Taking a look at some of confused letters, like V and W in Figure, we see that the two letters are similar in shape. This provides us with some intuition about why the model might confuse these two letters, as a W looks like a repeated V.

Reflection

Key Learnings

• Train Early and Often: Due to the long training time of our model, it became cumbersome to test various hyperparameters, architectures, image filtering, etc. In future projects, training earlier can identify such issues and more time can be dedicated to training





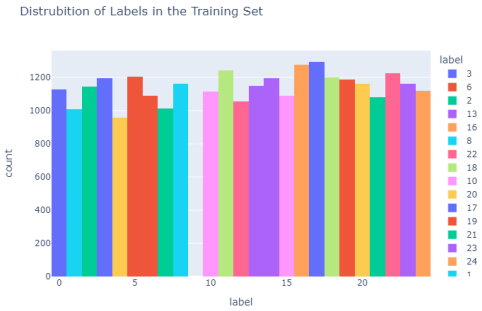
(a) Training and Validation Accuracy (b) Training and Validation Loss

Figure: Training and Validation Performance of Model Trained on Filtered Images

• Experiment With Optimizers: Due to our initial difficulties with Adam, we abandoned it for he slower SGD. However, further testing with Adam and varying our hyperparameters allowed Adam to converge. The lesson to take away is to try various optimizers early on to establish which ones converge the quickest for faster training.

• Amount of Data: As shown in Section 5.1, increasing the training set size leads to drastic improvement in model performance. This likely increases the robustness of the model through learning more of the possible sample space.

Results



	precision	recall	f1-score	support
0	1.00	1.00	1.00	331
1	1.00	1.00	1.00	432
2	1.00	1.00	1.00	310
3	1.00	1.00	1.00	245
4	0.99	1.00	1.00	498
5	1.00	1.00	1.00	247
6	0.95	1.00	0.97	348
7	1.00	0.95	0.98	436
8	1.00	1.00	1.00	288
10	0.99	1.00	1.00	331
11	1.00	1.00	1.00	209
12	1.00	1.00	1.00	394
13	1.00	1.00	1.00	291
14	1.00	1.00	1.00	246
15	1.00	1.00	1.00	347
16	1.00	1.00	1.00	164
17	0.99	0.97	0.98	144
18	1.00	0.99	0.99	246
19	1.00	1.00	1.00	248
20	1.00	0.99	1.00	266
21	0.99	1.00	0.99	346
22	1.00	1.00	1.00	206
23	1.00	1.00	1.00	267
24	1.00	1.00	1.00	332
accuracy			1.00	7172
macro avg	1.00	1.00	1.00	7172
weighted avg	1.00	1.00	1.00	7172

Brief Thoughts on Results

We were able to get an accuracy of 100% according to the classification report. The biggest take away for me was that the Batch Normalization and Dropout layers really helped with increasing the accuracy. The Dropout layers seemed to really smooth out the training results as well.

Conclusions

- **Recap**

We were able to take images of the American Sign Language alphabet and use a CNN model to learn the alphabet and make predictions on new images. When training our CNN model, the training results were 99%

accuracy but the validation results had an accuracy of 100%. When applying the CNN model on our test images we got the result of 100% accuracy.

Future Work

The next step would be to apply this model to live video feed to detect the American Sign Language alphabet within the video and translate each sign on the screen in real time.

Future Steps

• Use Dynamic Loading for Dataset: Our original dataset was quite large and is impossible to use without a server with a lot of RAM and disk space. A possible solution is to split the file names into training, validation, and test sets and dynamically loading images in the Dataset class. Using such a loading technique would allow us to train the model on more samples in the dataset.

References

[1] Farnaz D. Notash and Elahe Elhamki. “Comparing loneliness, depres sion and stress in students with hearing impaired and normal students studying in secondary schools of Tabriz”. In: International Journal of Humanities and Cultural Studies February 2016 Special Issue (2016). issn: 2356-5926.

[2] “The Cognitive, Psychological and Cultural Impact of Communication Barrier on Deaf Adults”. In: Journal of Communication Disorders, Deaf Studies Hearing Aids 4 (2 2016). doi: 10.4172/2375-4427.1000164.

[3] Akash. SL Alphabet. url: https://www.kaggle.com/grassknoted/asl-alphabet. (accessed: 24.10.2018).

[4] Vivek Bheda and Dianna Radpour. “Using Deep Convolutional Networks for Gesture Recognition in Sign Language”. In: CoRR abs/1710.06836 (2017). arXiv: 1710.06836. url: http://arxiv.org/abs/1710. 06836