American Sign Language Classification

Anthony An[1] and Emily Haigh[1]

[1]Computer Science, Penn State Harrisburg, Penn State University, West Harrisburg Pike, Middletown, 17057, PA, USA.

Abstract

In our American Sign Language classification project, we used Python to make a convolutional neural network to classify images of the 26 letters of the alphabet in American Sign Language. We trained multiple models with 9 activation functions, 8 optimizer functions, and 12 loss functions to find the most accurate and efficient model. With stochastic gradient descent optimizer, hyperbolic tangent hidden layer activation function, sigmoid output layer activation function, and sparse categorical cross entropy loss function, our final model had a testing accuracy of 91.8% after 3000 epochs.

Keywords: Machine Learning, Deep Learning, CNN, Classification, American Sign Language

Contents

1 Introduction

American Sign Language (ASL) is a natural and complete language, which contains different grammar from English. In the United States, it is assumed that there are about a million people who are functionally deaf[1]. Those who use ASL as their main language still have problems communicating with people who are not familiar with it. If ASL can be converted into another form immediately without additional cost, it will be highly applicable in various fields.

Z. A. Ansari and G. Harit[2] classified static sign language images using an unsupervised learning algorithm. They used 3 color channel images in Standard Definition resolution and the scale-invariant feature transform to extract features. However, the accuracy of recognizing 13 signs was around 90%. Also, J. Rekha, J. Bhattacharya, and S. Majumder[3] performed a similar classification using non-linear support vector machines. The accuracy of recognizing 23 signs was around 91%.

As the previous works indicate, the methodologies based on traditional classification techniques cannot be considered very accurate. To increase the accuracy of the classification of images, using supervised learning algorithms is recommended since it produces more accurate results. One of the most powerful supervised learning algorithms is deep learning algorithms, which can handle large numbers of features to find useful data. Hence, by applying Machine Learning to ASL, our goal of this project was to create a convolutional neural network to accurately classify images of the ASL alphabet. This is an important problem to solve because it could lead to developments in technology that allow deaf and hearing people to communicate more effectively.

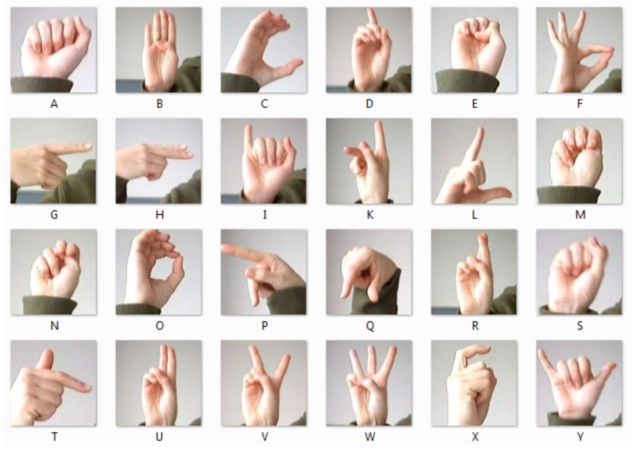

Fig. 1: American sign language alphabets

1.1 Contribution

- Gathering and pre-processing the dataset.
- Creating a convolutional neural network.
- Training models with 9 activation, 8 optimizer, and 12 loss functions.
- Testing the models.

2 Materials and Methods

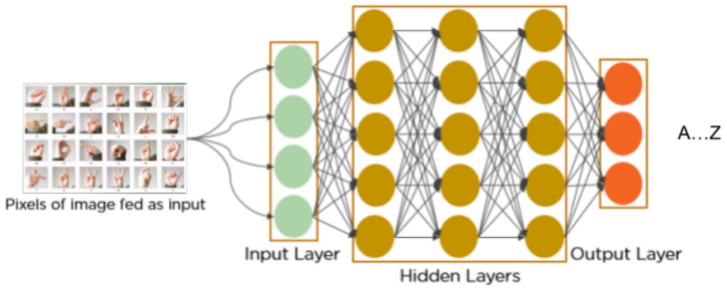

Fig. 2: Abstract structure of our model

Our model is trained with a bunch of gesture images. The trained model takes an image as an input and gives a label as an output.

2.1 Dataset Description

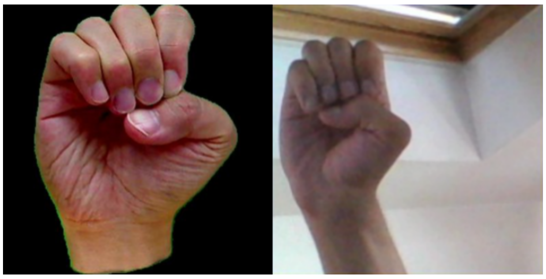

Fig. 3: Images of E in the dataset

Our dataset is a combination of Dataset1 and Dataset2 found on Kaggle. We used the two different datasets because they contain different hand shapes, backgrounds, and brightness. We decided to resize all the images to be 200 pixels by 200 pixels. This is because smaller images can be processed faster and have about the same quality as the original larger image. We also converted the RGB values of each pixel to grayscale using equation (1) so that fewer calculations were required. The equation is normally used to calculate the brightness based on CCIR 601, which is the digital SD format. By using the equation, we were able to convert the color images to the grayscale images without losing important features of each image. Lastly, for the file management convenience purpose, we converted images to CSV files. Therefore, each image is represented as a list of 40,000 grayscale values.

$$Y_{linear} = R_{linear} * 0.299 + G_{linear} * 0.587 + B_{linear} * 0.114 \tag{1}$$

2.2 Methodology

We trained models, using our own program, with different combinations of the following 9 activation functions, 8 optimizer functions, and 12 loss functions.

- Activation functions: relu, sigmoid, softmax, softplus, softsign, tanh, SELU (scaled exponential linear unit), ELU (exponential linear unit), and exponential.
- Optimizer functions: stochastic gradient descent, RMSProp (root mean squared propagation), adam, adadelta, adagrad, adamax, nadam, and FTRL (follow the regularized leader).
- Loss functions: sparse categorical cross entropy, poisson, KL divergence, mean squared error, mean absolute error, mean absolute percentage error, mean squared logarithmic error, huber, log cosh, hinge, squared hinge, categorical hinge.

It was clear from early tests that the sparse categorical cross entropy loss function yielded better results than any other loss function. So next, we tried combinations of the 9 activation functions, 8 optimizer functions, and the 1 loss function with 1,000 epochs in each model. We chose 1,000 epochs due to time constraints. Then, we decided to continue training the models that yielded greater than 70% testing accuracy. At this point, we decided to have different activation functions for the hidden layers and the output layers. Based on the usage, we carefully chose new functions from the papers written by J. Gu et al.[4] and S. Bera and V. Shrivastava[5]. That left us with 3 activation functions for the hidden layers, 3 activation functions for the output layers, 5 optimization functions, and the same sparse categorical cross entropy loss function.

- Hidden activation functions: relu, sigmoid, and tanh.
- Output activation functions: linear, sigmoid, and softmax.
- Optimization functions: stochastic gradient descent, rmsprop, adam, adadelta, adagrad.

Then, the models with combinations of these functions were each trained for 3,000 epochs.

2.3 Performance Evaluation

We gathered 14,820 raw images and split them into the training/validation dataset and test dataset with a ratio of 8:2. Then, we split the training/-validation dataset into the training dataset which is used to extract features and train the model, and the validation dataset which is used to tune the hyperparameters of our deep learning model with a ratio of 8:2.

3 Experimental Analysis

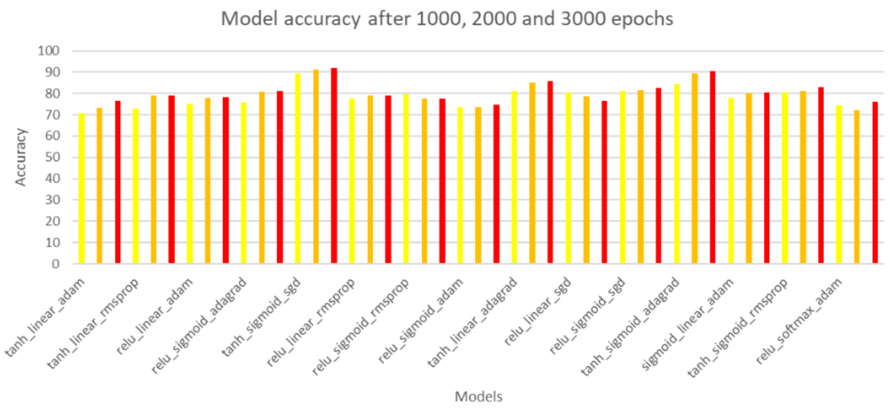

Fig. 4: Model accuracy after 1,000, 2,000, and 3,000 epochs

In Fig. 4, the yellow bar shows the accuracy after 1,000 epochs. Orange is after 2,000 epochs and red is after 3,000 epochs. This graph only includes models that had a testing accuracy of greater than 70% after 1,000 epochs. The models are named x_y_z, x represents the hidden layer activation function, y represents the output layer activation function, and z represents the optimizer. Each model uses sparse categorical cross entropy loss.

Due to the limitation of time and hardware problems, we were only able to reach 3,000 epochs at the end. The model with hyperbolic tangent and sigmoid function with stochastic gradient descent optimizer showed the highest accuracy rate when it was tested with 2,736 images. We could not find a specific pattern that increases the overall accuracy, but the optimizer function seems to be playing an important role in each model as the models with the highest accuracy rate use sgd or adagrad as an optimizer function.

4 Conclusion and Future Work

In conclusion, we were able to achieve 91.8% testing accuracy. Our biggest limitation during this project was computer power. The sunlab computers were

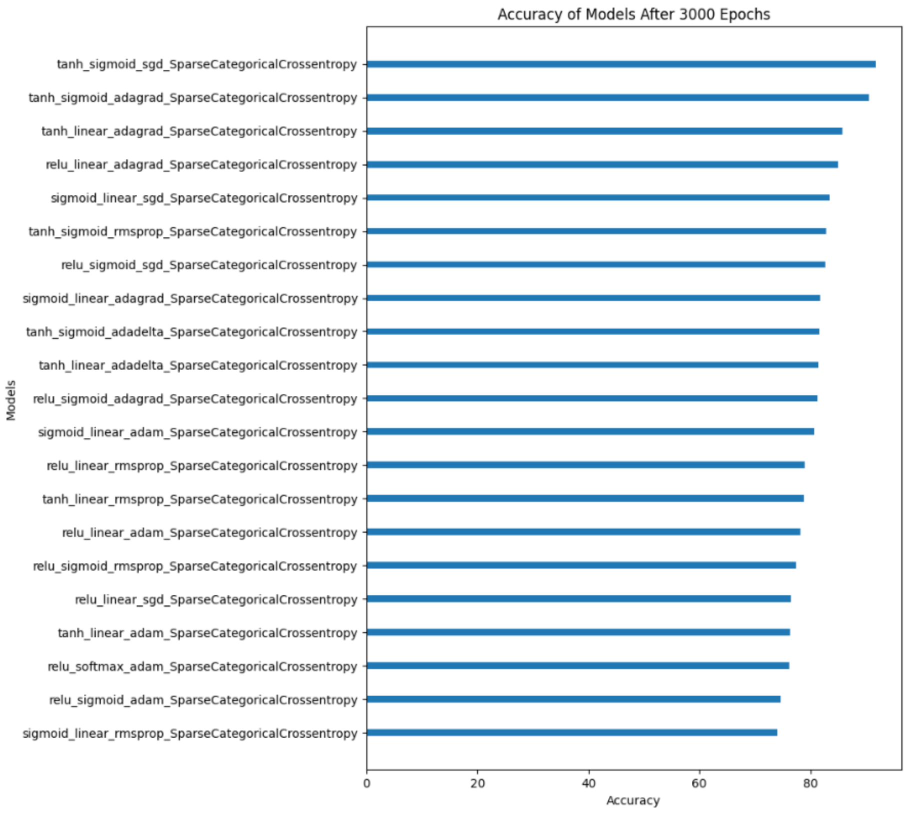

Fig. 5: Models sorted by testing accuracy after 3000 epochs

slow and it took a few days to train each model. We were only able to train our models for 3,000 epochs. More powerful computers would have saved us a lot of time. If the time permitted, we would finish training our models and directly compare the accuracy with the test results from other papers.

In the future, we would like to expand the dataset to include high resolution images of many different kinds of hands. This could also be extended to the whole language, not just the alphabet. Also, we would like to convert dynamic sign languages by implementing a real-time recognition program with additional functionalities, such as tracking and segmenting hands and extracting related features from images with an arbitrary background.

References

[1] Mitchell, R.: How many deaf people are there in the united states? estimates from the survey of income and program participation. Journal of deaf studies and deaf education **11**, 112–9 (2006). https://doi.org/10.1093/deafed/enj004

[2] Zafar, A., HARIT, G.: Nearest neighbour classification of indian sign language gestures using kinect camera. Sadhana **41** (2016). https://doi.org/10.1007/s12046-015-0405-3

[3] Rekha, J., Bhattacharya, J., Majumder, S.: Shape, texture and local movement hand gesture features for indian sign language recognition. In: 3rd International Conference on Trendz in Information Sciences Computing (TISC2011), pp. 30–35 (2011). https://doi.org/10.1109/TISC.2011.6169079

[4] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., Chen, T.: Recent advances in convolutional neural networks. Pattern Recognition **77**, 354–377 (2018). https://doi.org/10.1016/j.patcog.2017.10.013

[5] Bera, S., Shrivastava, V.: Analysis of various optimizers on deep convolutional neural network model in the application of hyperspectral remote sensing image classification. International Journal of Remote Sensing **41**, 2664–2683 (2020). https://doi.org/10.1080/01431161.2019.1694725