Image Classification of American Sign Language using CNN

Anshul Rao
Khoury College of Computer Sciences
Northeastern University
Boston, MA 02115
rao.ans@northeastern.edu

Abstract

Convolutional neural networks were utilized to classify images of American Sign Language (ASL) letters using different techniques.
I built and trained a CNN with three convolutional layers from scratch, and I also trained another one using a pre-trained model GoogLeNet. All of the models had accuracy levels exceeding 98 percent. Additionally, the incorrectly classified images were examined, and as a final step, I developed a straightforward application that made use of the model to classify motions in real-time using webcam video.

Introduction

As part of this project, I classified images of American Sign Language (ASL) letters. There are many people with impaired hearing and speaking disabilities who rely on sign language for communication but not enough human translators. Hence, I wanted to explore this domain and the idea is that building a model that is able to classify ASL images is one step towards understanding and possibly building a system that converts sign language to speech or text.

I built and trained a Convolutional Neural Network (CNN) from scratch using three convolutional layers and I also trained another one using a pre-trained model GoogLeNet (i.e., do transfer learning). For transfer learning, I explored both approaches, namely fine-tuning and feature extraction. CNNs have been extremely successful in image recognition and classification problems. Hence, it was the apt choice. The convolutional layers reduce the high dimensionality of images without losing its information.

I also used regularization techniques to avoid overfitting. In CNN with three convolutional layers I added a dropout layer and in the ones created using the pre-trained model I used weight decay.

For results, I used the classic classification metric accuracy and plotted curves for training and test losses.

Methodology

The dataset[1] has 87k images belonging to 29 classes which are basically the 26 letters in the English alphabet and ‘space’, ‘nothing’ and ‘delete’. Each image has dimensions 200 x 200 and has three channels (is colored). Refer to the figure below showing some of the images from the dataset.

I conducted my work using JupyterLab Notebook [Custom Anaconda Environment] on the Discovery cluster.

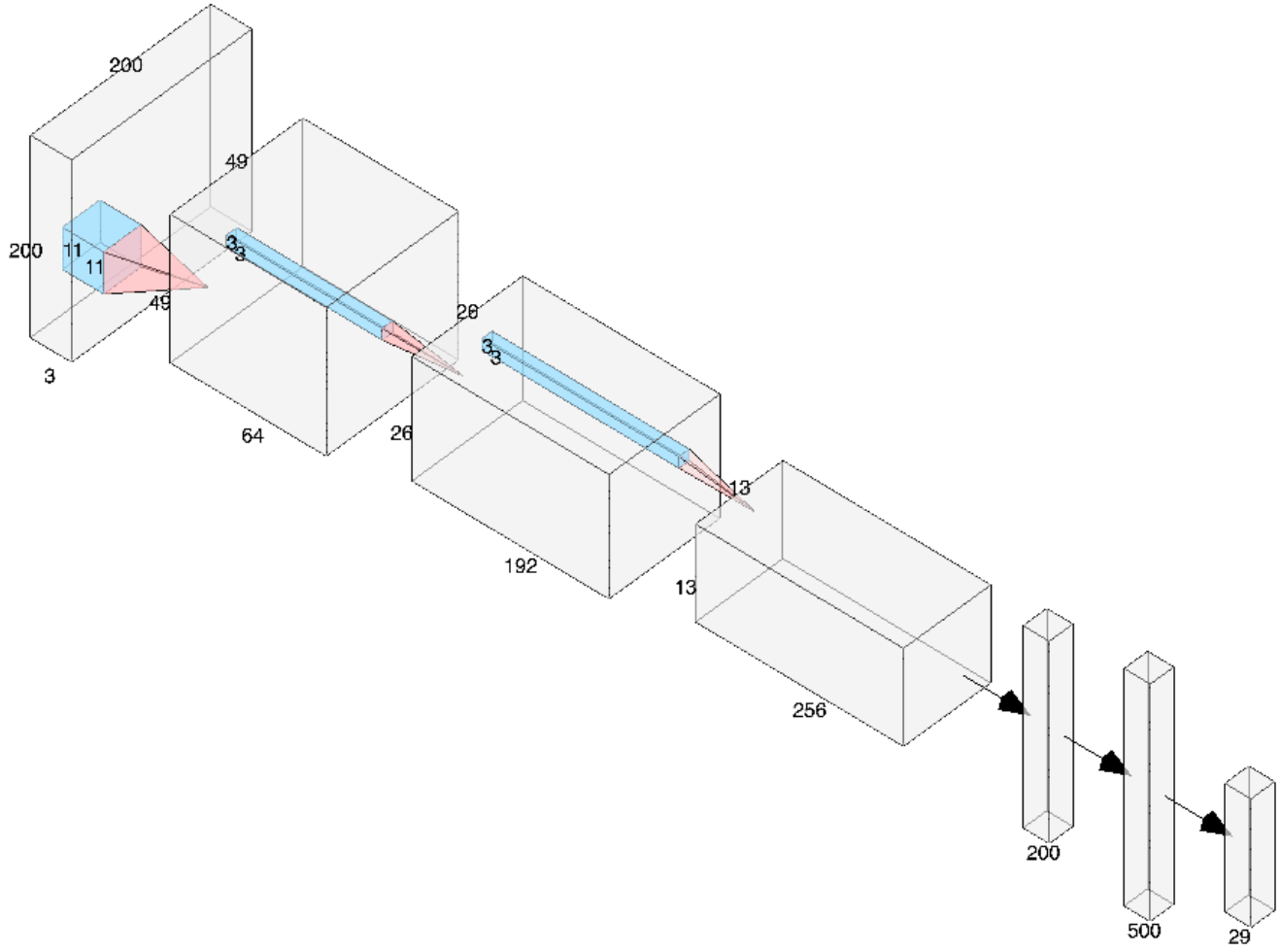
GPU	v100-smx2
CUDA Module	cuda/11.4
Conda Module	anaconda3/2022.01

Model 1: CNN (Scratch):

The first model is a convolutional neural network that was built from the ground up using three convolutional layers. Cross-entropy loss was calculated and stochastic gradient descent was used for optimization. To prevent overfitting, a dropout layer was applied as a regularization technique. The model was trained for 10 epochs and there were 2,535,133 trainable parameters in total.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 49, 49]          23,296
              ReLU-2          [-1, 64, 49, 49]               0
         MaxPool2d-3          [-1, 64, 24, 24]               0
            Conv2d-4         [-1, 192, 26, 26]         110,784
              ReLU-5         [-1, 192, 26, 26]               0
         MaxPool2d-6         [-1, 192, 13, 13]               0
            Conv2d-7         [-1, 256, 13, 13]         442,624
```

```
                ReLU-8          [-1, 256, 13, 13]               0
           MaxPool2d-9            [-1, 256, 6, 6]               0
            Linear-10                  [-1, 200]       1,843,400
              ReLU-11                  [-1, 200]               0
           Dropout-12                  [-1, 200]               0
            Linear-13                  [-1, 500]         100,500
              ReLU-14                  [-1, 500]               0
            Linear-15                   [-1, 29]          14,529
================================================================
```



The diagram was constructed using NN-SVG[2].

Model 2: GoogLeNet (Feature extraction)

In this case, I used GoogLeNet, which is a pretrained convolutional neural network that is 22 layers deep. To adapt this network to the specific use case, I added a fully-connected layer with

29 units to the end of the network. I calculated cross-entropy loss and used the Adam optimizer to train the model. To prevent overfitting, I applied weight decay as a regularization technique. I used the feature extraction approach to transfer learning and froze all but the last three convolutional layers in the pretrained model. The model was trained for 30 epochs and had a total of 232,061 trainable parameters.

Model 3: GoogLeNet (Fine-tuning)

In this case, I again used GoogleNet but took a different approach to transfer learning. I employed the fine-tuning approach, which involved training all of the model's parameters. To adapt the network to our specific use case, I added a fully-connected layer with 29 units to the end of the network, which allowed it to classify images into 29 different classes. I calculated cross-entropy loss and used the Adam optimizer to train the model. To prevent overfitting, I applied weight decay as a regularization technique. The model was trained for 30 epochs and had a total of 5,629,629 trainable parameters.

Lastly, I also created a small software that used the model to classify motions in real-time using webcam video.

Results

The accuracy of the three models are as follows:

Model	**Epochs**	**Trainable Parameters**	**Accuracy (Test)**
CNN (Scratch)	10	2,535,133	98.50
GoogLeNet (Feature extraction)	30	232,061	99.47
GoogLeNet (Fine-tuning)	30	5,629,629	99.97

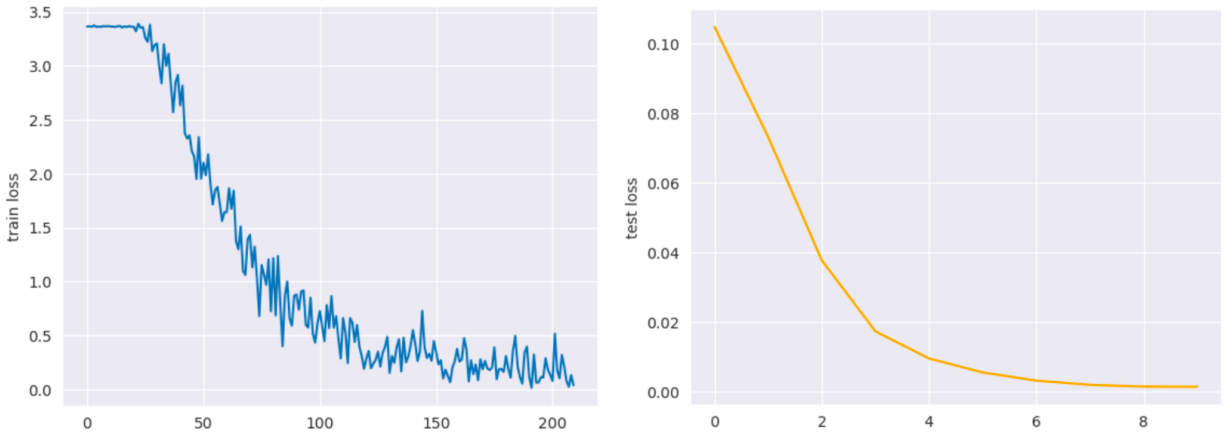
As can be seen, all of the models had an accuracy that exceeded 98 percent, which is quite good. The model built from scratch was trained for just 10 epochs still it had an accuracy of 98.5. Feature extraction did well with 99.47 accuracy using only 232,061 trainable parameters. It was comparatively faster than fine-tuning too. But fine-tuning, though it took longer, gave an accuracy of 99.97 which was brilliant in my opinion. Note that all the accuracies mentioned are calculated on the test dataset.

I also spent some time analyzing the incorrect predictions made by the model and tried to identify the reasons for the inaccuracies. Refer to the image below for 20 incorrect predictions.

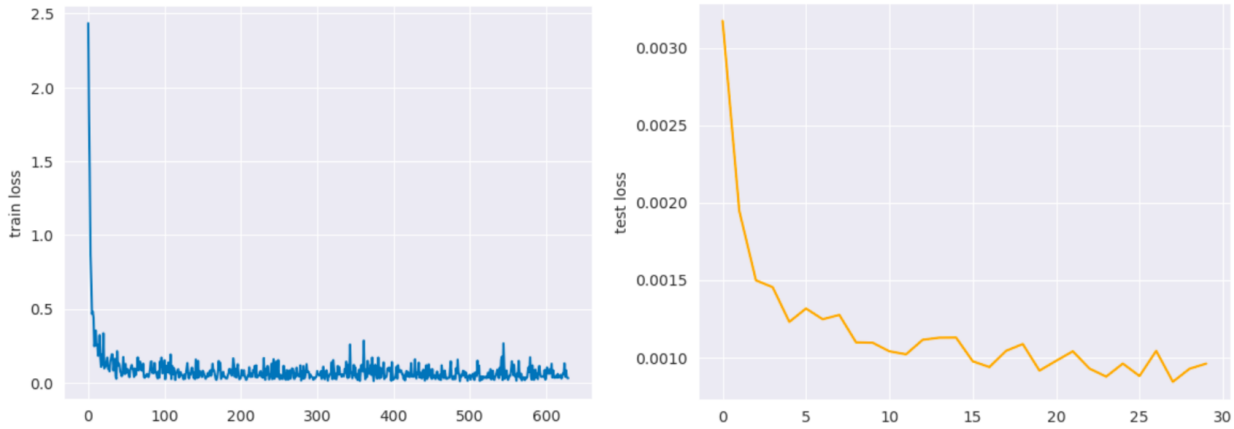
Below are some of the reasons I identified for the misclassifications:-

- Many J gestures have been classified as I because they are practically similar signs with J having motion. (J and Z are two alphabets that have motion). Refer: [1], [2], [3], [12], [16] and [18].
- U and R are also similar gestures with R having the two fingers crossed. So when it is not very clear to see that the fingers are crossed, they are classified as U. Refer [13] and [20].
- Otherwise, some images are cropped and unclear. For example, [5], [6], [7], [8], [10], [11] and [14] are some cropped images and [11] is unclear.

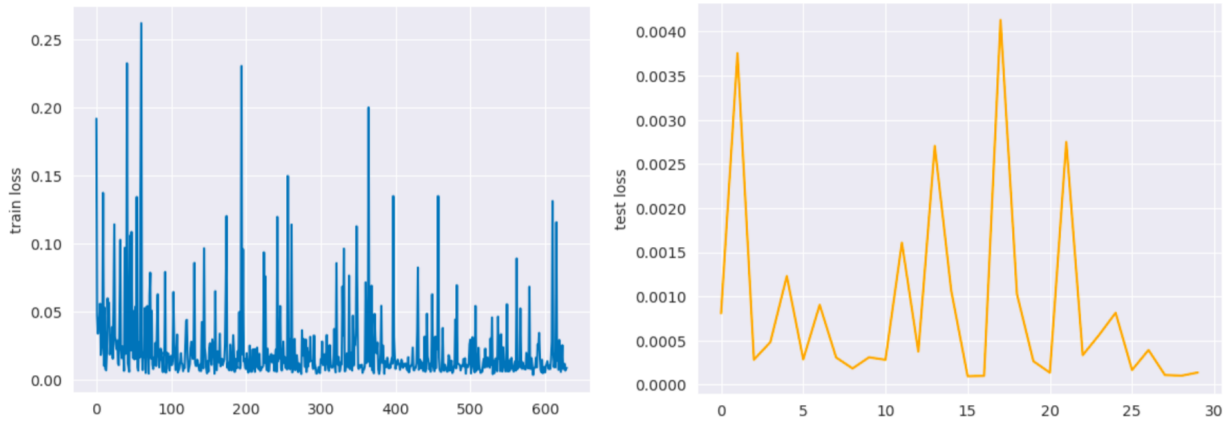
The plots for losses in all the three models are as below:-



CNN (Scratch)



GoogLeNet (Feature extraction)



GoogLeNet (Fine-tuning)

While experimenting with the real-time video classification application, I noticed that the classifier performed more accurately when I brought my entire palm into the frame and moved it closer to the camera. Additionally, if there were any objects in the frame other than the intended gesture, the classifier performed poorly, which is to be expected.

Discussion

The results I obtained in this project are promising. Many of the existing classifiers have an accuracy of 99 percent for ASL classification, so my work is comparable to the current benchmarks in this area.

When I started on this project, the first model itself gave an accuracy as high as 98 percent and so based on the high accuracy, I decided not to perform any parameter tuning. The high accuracy also sort of made it difficult to compare the performance of the models rigorously, as they all performed pretty well. But at the same time, although marginally different, the accuracies highlight that fine-tuning performed the best, as expected. This is because it used a pre-trained model with 22 convolutional layers, and all of its parameters were re-trained and rather than using random initialization, the pre-trained values were used to initialize the parameters.

There are several potential directions for future work in this area. One potential direction would be to explore other types of neural network architectures and compare their performance with CNNs on the ASL dataset. Additionally, it would be interesting to investigate the use of generative models such as generative adversarial networks (GANs) for synthesizing additional ASL images to augment the training dataset and improve the performance of the models. Another interesting direction for future work would be to evaluate the performance of the CNN models on other datasets of ASL images and compare the results. This could provide valuable insights into the generalizability of the models and help identify any potential limitations of the approach.

Conclusion

In conclusion, I would say convolutional neural networks (CNNs) are an effective method for classifying American Sign Language (ASL) images. In this project, I trained and evaluated several CNN models on the dataset of ASL images and achieved high accuracy rates. I learned both how to build and train a model from scratch and how to use transfer learning. During my inspection of incorrect predictions and real-time classification of video capture, I realized that the quality of the input data has a significant impact on the performance of the classifier. For

example, if the image is unclear or cropped, the classifier may perform poorly. This emphasizes the importance of ensuring high-quality data when training and evaluating image classification models.

References

1. Kaggle: ASL Alphabet, https://www.kaggle.com/datasets/grassknoted/asl-alphabet
2. NN-SVG, http://alexlenail.me/NN-SVG/AlexNet.html
3. Real-time American Sign Language Recognition with Convolutional Neural Networks, http://cs231n.stanford.edu/reports/2016/pdfs/214_Report.pdf

Appendix

- CODE: https://github.com/anshulrao/asl-image-classifier
- Screen capture of the video classification application:-