Computer Vision for Reading American Sign Language

Project Report

EE 541: A Computational Introduction to Deep Learning
Prof. Brandon Franzke

Aravindh Soundararajan (USC ID: 4602478722)
aravindh@usc.edu
Varsha Bhat (USC ID: 2522688802)
varshava@usc.edu

Abstract

American Sign Language is the primary communication tool for many Americans who are hard of hearing. This language is expressed by hand movements and facial expressions. With this project, we aim to build deep learning models to read the American Sign Language. Our ultimate goal is to increase the accuracy of our model by testing various deep learning frameworks, varying the hyper-parameters, and experimenting with various layers. We finally integrate these models with our OpenCV application to identify the characters in real-time and display the predicted class.

Table of Contents

1. Introduction

We aim to build an application that can read the American Sign Language in Real-Time. This system will use a Deep Neural Network to identify each character in the American Sign Language and display the text on the screen. This way, any person not familiar with ASL can use this application to communicate with people who are hard of hearing. To implement the model, we identified many deep neural network architectures, i.e., baseline deep neural network, convolutional neural network, ResNet9, and the DenseNet, and trained them. We experimented with various combinations of the parameters, the hyper-parameters, the number of layers in the neural network, etc. Finally, we used the optimal combination to build the application.

2. Dataset

The dataset is 1.03 GB. It is split into a training and test set, with 3000 images for each character (26 alphabets, SPACE, DELETE, and NOTHING) in the training set (total of 87,000 images), and 28 images in the test set, all in JPG format. Each image consists of 200x200 RGB pixels. The dataset can be found here at Kaggle.
To accurately identify the test error and accuracy, we captured additional data for testing using a python application. The application captures 200x200 RGB pixel images, each image in PNG format. We captured 10 additional images for each character (a total of 290 additional test images) using this application.
Additionally, to reduce processing time, we downsize these images to 32 x 32 RGB.

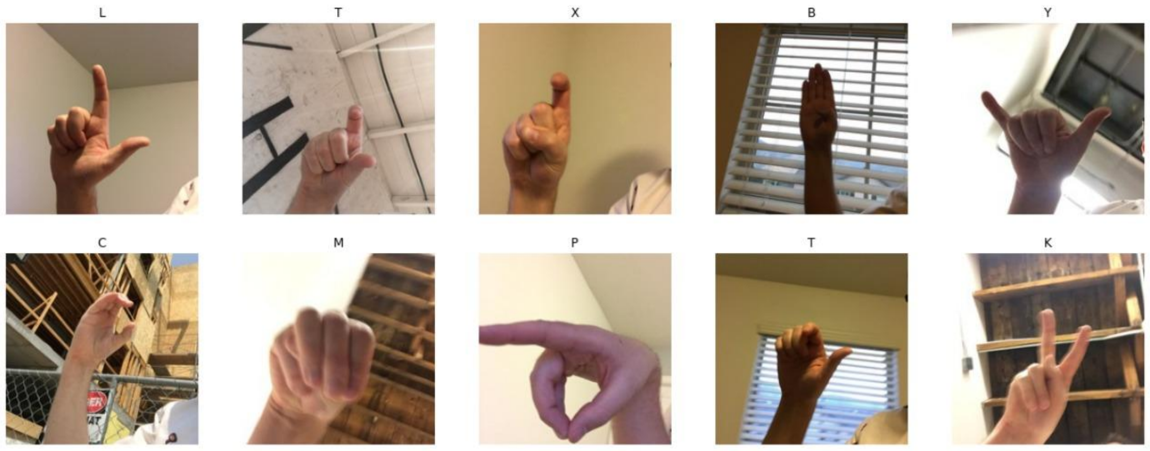


Figure 2.1: American Sign Language Characters with their corresponding English Labels

3. Related Work

3.1 Convolutional Neural Networks for ASL [1]

Facilitating communication between people who are hard of hearing and the general public has always remained a challenge. This paper explores a translator using Convolutional Neural Networks. We aim to build on the approach and create a fully functioning sign language interpreter capable of recognizing all 29 classes in the Kaggle dataset.

3.2 Deep Residual Learning for Image Recognition [2]

As we go deeper, it gets harder to train neural networks. Residual Networks overcome this challenge by providing a framework that allows us to go deeper, while also solving the vanishing/exploding gradient problem. This paper demonstrates Deep Residual Learning with its architecture, and how it can be used to train the ImageNet and CIFAR-10 database.

4. Architecture

We trained 4 models for this dataset, a baseline Deep Neural Network, a Convolutional Deep Neural Network, ResNet9, and DenseNet121. A detailed analysis of these architectures is described below.

4.1 Deep Neural Network

In order to establish a baseline for this dataset, we build a deep neural network, keeping the number of layers and other optimizations as low as possible, i.e., two layers (including input and output layers) and 1 ReLU activation function. The architecture of this model is described in Figure 4.1.

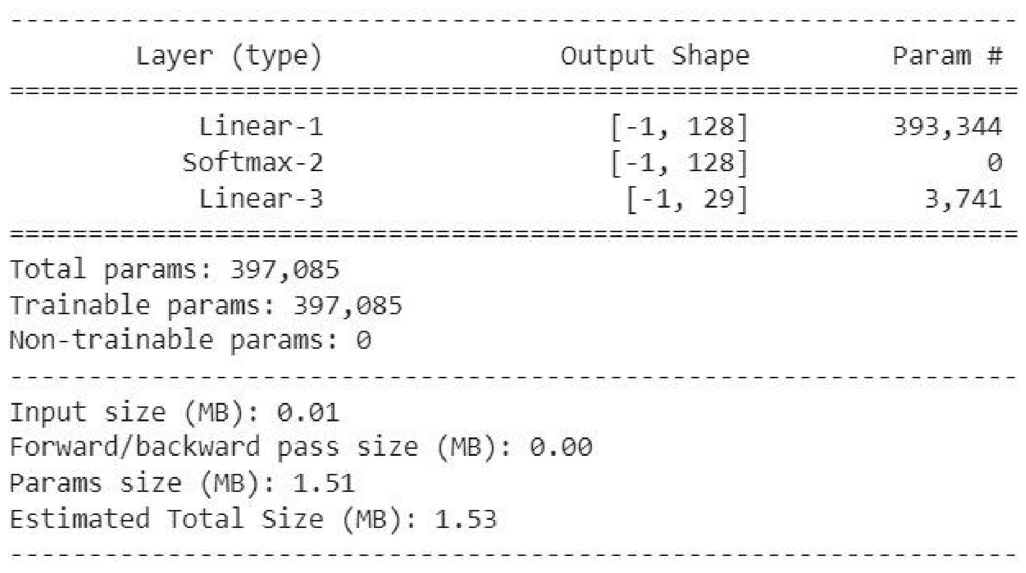

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 128]         393,344
           Softmax-2                  [-1, 128]               0
            Linear-3                   [-1, 29]           3,741
================================================================
Total params: 397,085
Trainable params: 397,085
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.00
Params size (MB): 1.51
Estimated Total Size (MB): 1.53
----------------------------------------------------------------
```

Figure 4.1: Architecture of our Deep Neural Network

4.2 Convolutional Neural Network [3] [7]

The goal of our problem is to map the input images to a character in ASL. A convolutional Neural Network is a good fit for a problem of this nature. For the ASL dataset, we use max-pooling with padding, sigmoid and ReLU activation functions, and dropouts. The structure of our architecture is shown in Figure 4.3.

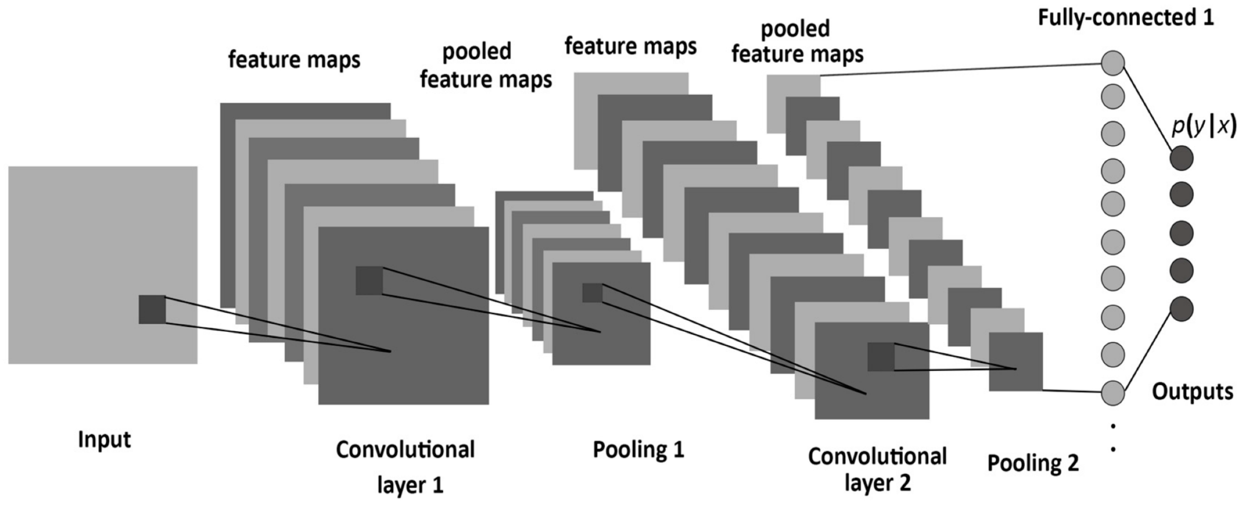
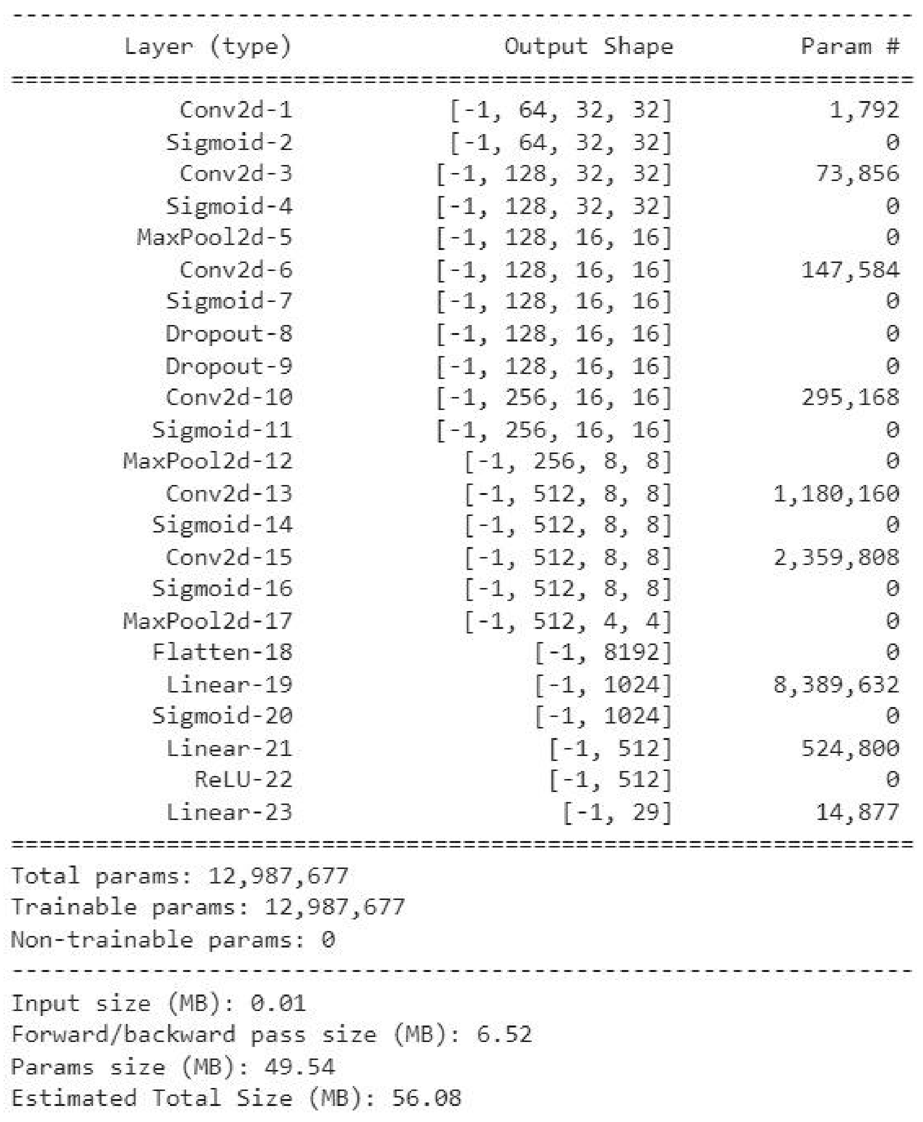


Figure 4.2 [3]: A generic CNN architecture showing the input-output mapping and various layers that can be used



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
           Sigmoid-2           [-1, 64, 32, 32]               0
            Conv2d-3          [-1, 128, 32, 32]          73,856
           Sigmoid-4          [-1, 128, 32, 32]               0
         MaxPool2d-5          [-1, 128, 16, 16]               0
            Conv2d-6          [-1, 128, 16, 16]         147,584
           Sigmoid-7          [-1, 128, 16, 16]               0
           Dropout-8          [-1, 128, 16, 16]               0
           Dropout-9          [-1, 128, 16, 16]               0
           Conv2d-10          [-1, 256, 16, 16]         295,168
          Sigmoid-11          [-1, 256, 16, 16]               0
        MaxPool2d-12            [-1, 256, 8, 8]               0
           Conv2d-13            [-1, 512, 8, 8]       1,180,160
          Sigmoid-14            [-1, 512, 8, 8]               0
           Conv2d-15            [-1, 512, 8, 8]       2,359,808
          Sigmoid-16            [-1, 512, 8, 8]               0
        MaxPool2d-17            [-1, 512, 4, 4]               0
          Flatten-18                 [-1, 8192]               0
           Linear-19                 [-1, 1024]       8,389,632
          Sigmoid-20                 [-1, 1024]               0
           Linear-21                  [-1, 512]         524,800
             ReLU-22                  [-1, 512]               0
           Linear-23                   [-1, 29]          14,877
================================================================
Total params: 12,987,677
Trainable params: 12,987,677
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 6.52
Params size (MB): 49.54
Estimated Total Size (MB): 56.08
```

Figure 4.3. Architecture of our Convolutional Neural Network

4.3 ResNet9 [4] [7]

The performance of a classic CNN does not increase when we add layers past a certain threshold. The ResNet architecture offers a solution and allows us to go deeper. It also solves the vanishing gradient problem by using skip connections. These skip connections are illustrated in Figure 4.4. The skip connections essentially skip learning for a few intermediate layers, and jump over. If any layer hurts the performance of the model, this layer will be skipped over with regularization. The ResNet9 is a variant of such Residual Neural Networks and is 9 layers deep. **We use this model to identify images in the real-time ASL character predictor using OpenCV.**

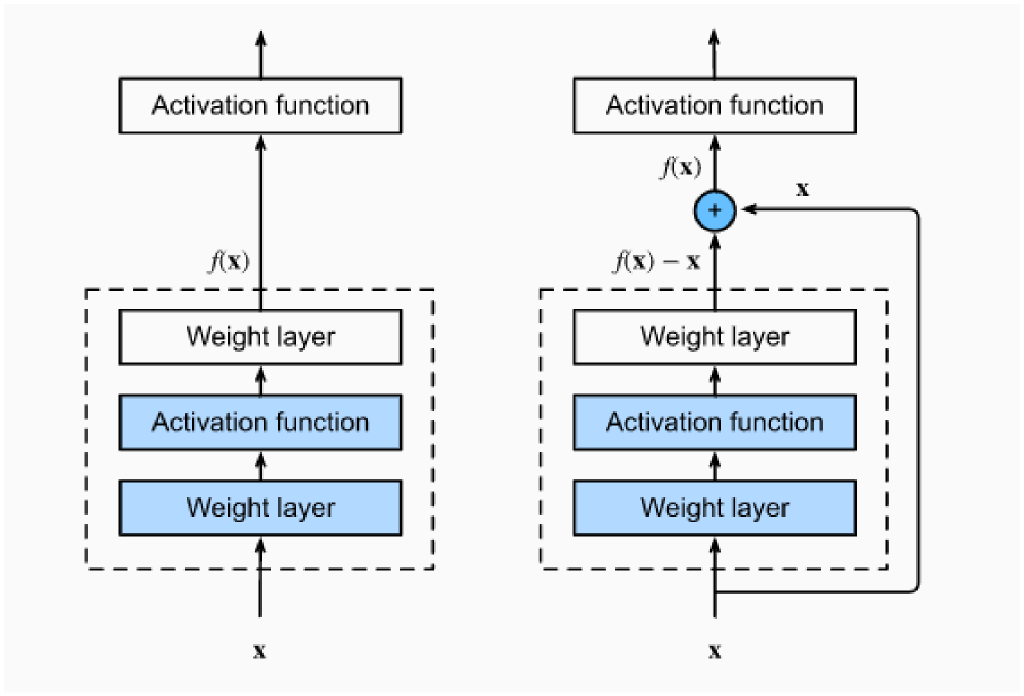


Figure 4.4 [4] Illustration of neural network layers without (left) and with (right) skip connections

4.4 DenseNet121 [5] [6]

This architecture allows the weights to be spread over multiple layers. Hence, the deeper layers can begin using relevant features early on. A detailed examination of this architecture can be found in this reference [5].

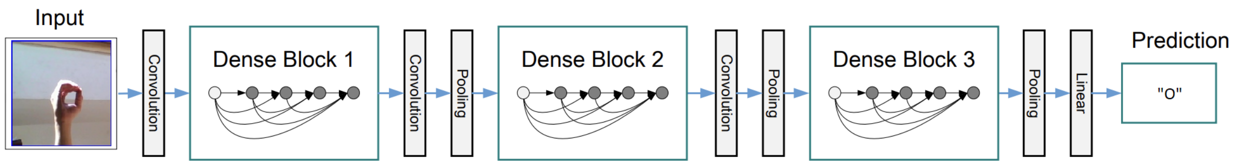


Figure 4.5 [6]: Illustration of a DenseNet Architecture with Dense Blocks

5. Training

The table below gives an overview of all the models used and the configurations used while training these models

Table 5.1	
Model	**Layers, Optimizers, and Loss Functions**
Baseline DNN	3 layers, Cross Entropy Loss, SGD Optimizer
CNN	23 layers, Cross Entropy Loss, SGD Optimizer
ResNet9	9 layers, Cross Entropy Loss, Stochastic Batch GD, SGD Optimizer
DenseNet	121 layers, Cross Entropy Loss, Adam Optimizer

Table 5.1: Table illustrating all models used for training the ASL along with their configurations

6. Results

Live Demo

The link to a real-time demonstration of our application can be found HERE.

Table 6.1 outlines the accuracies obtained for the ASL dataset during our exploratory analysis.

Table 6.1			
	Training Acc	**Validation Acc**	**Test Acc**
Baseline DNN	0.875	0.865	0.863
CNN	0.998	0.996	0.997
ResNet9	0.9998	1	0.9996
DenseNet	0.8262	0.8161	0.83088

Table 6.1: A comparison of Training, Validation, and Test Accuracies for the 4 models used to train the ASL dataset.

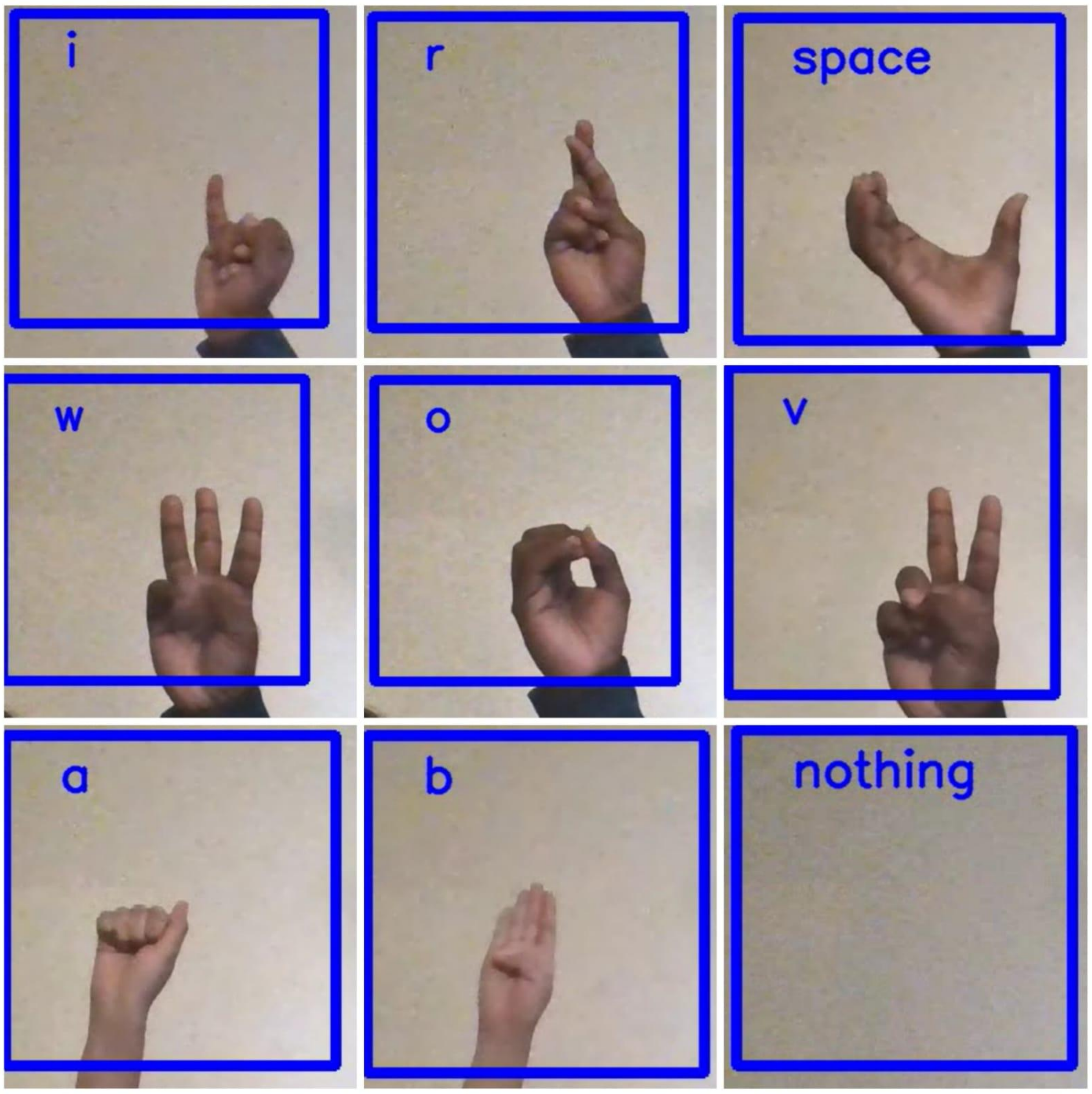


Figure 6.1: Real-Time outputs of correctly classified characters in the American Sign Language. The gestures along wit their predicted classes are shown in the above figure.

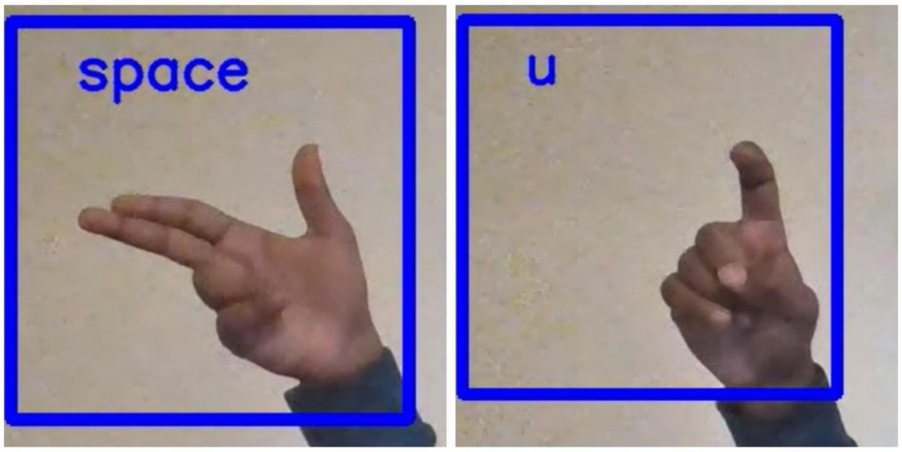


Figure 6.2: Real-Time outputs of incorrectly classified characters in the American Sign Language. The first character is the gesture for “H” and the second character is the gesture for “X”, but they are incorrectly classified as “space” and “u” respectively.

Learning Plots

The figures below illustrate the learning process by showing the losses, accuracies and confusion matrices for various models.

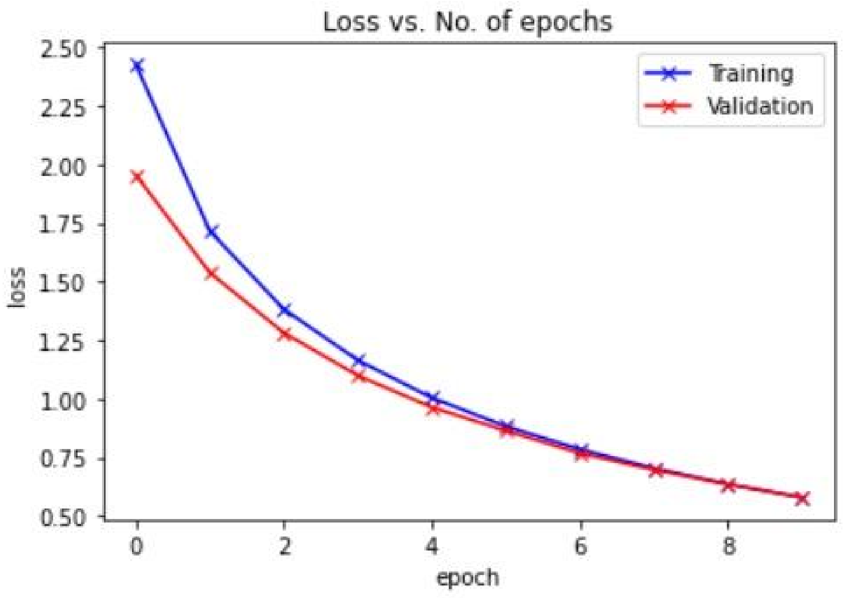


Figure 6.3: Training and Validation Loss vs No. of Epochs for the baseline Deep Neural Network

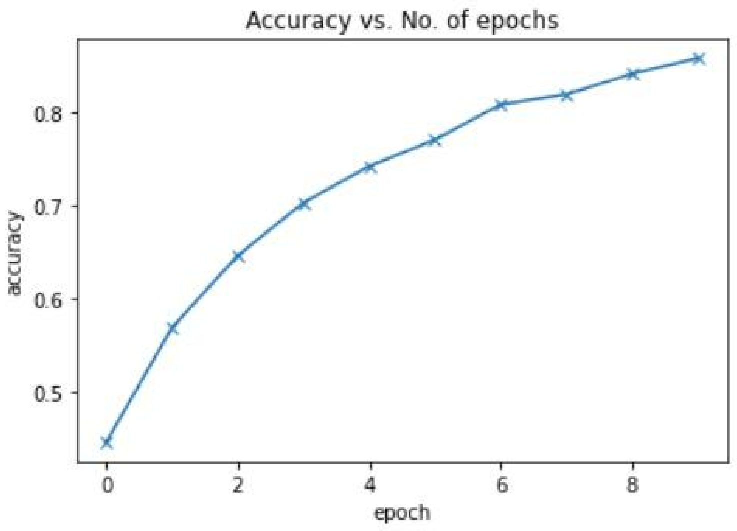


Figure 6.4: Training Accuracy vs No. of Epochs for the baseline Deep Neural Network

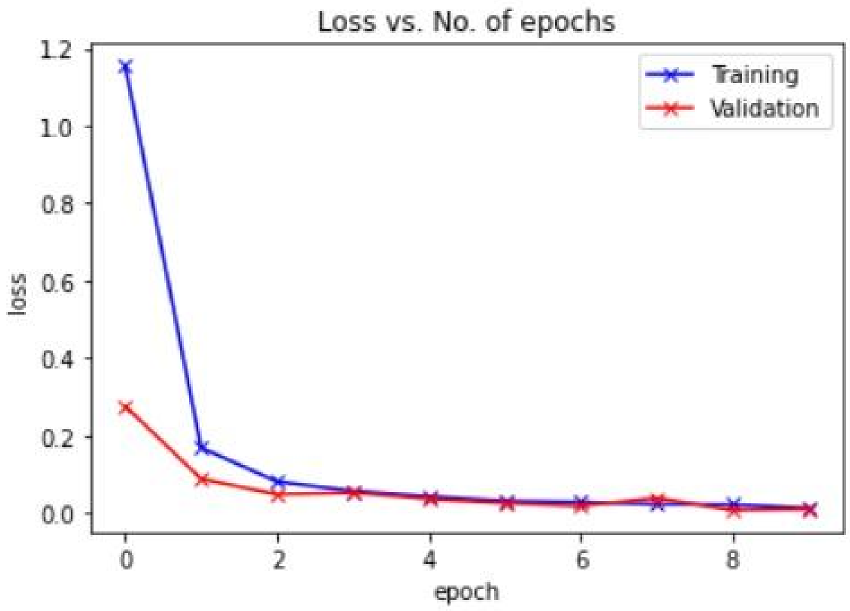


Figure 6.5 Training and Validation Loss vs No. of Epochs for the Convolutional Neural Network

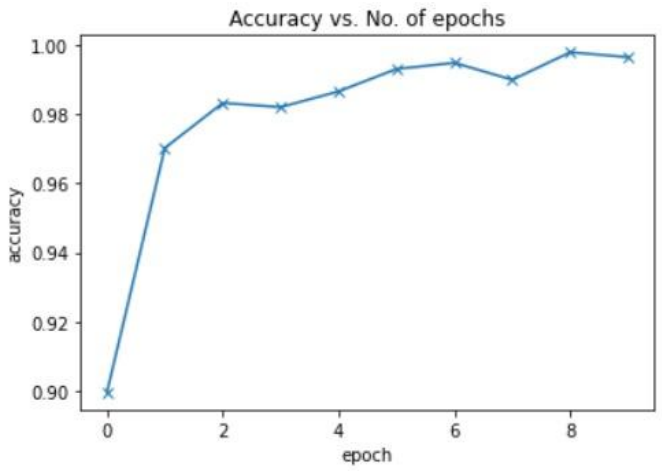


Figure 6.6 Training Accuracy vs No. of Epochs for the Convolutional Neural Network

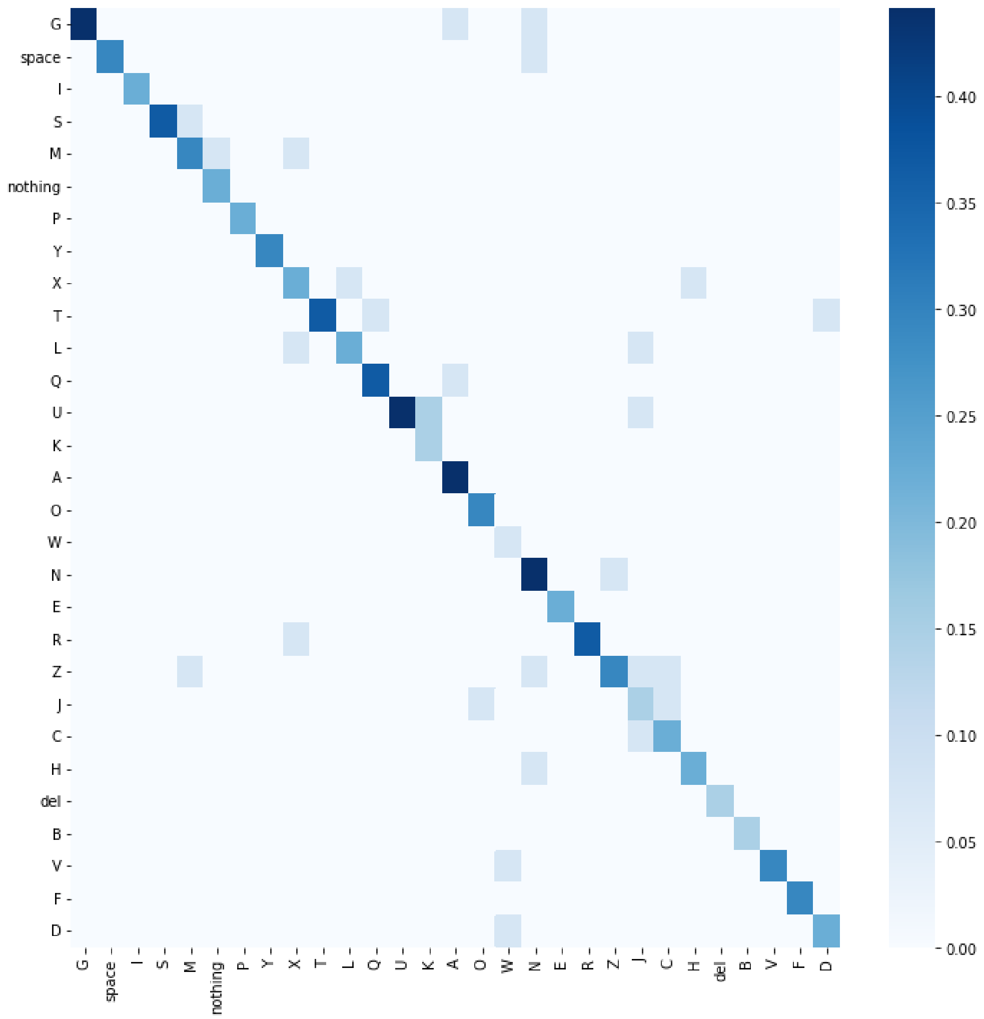


Figure 6.7: Confusion Matrix for DenseNet for the Test Set (gives a pictorial representation of prediction of classes and identifies which class is most likely to be confused with another).

7. Challenges

- The test dataset available at Kaggle is very limited. Manually compiling a diverse test dataset proved to be time-consuming.
- Adding too many layers without having a reasonable estimate could lead to vanishing/exploding gradient problems if not handled appropriately. This is a problem that is very difficult to diagnose during training and could lead to very bad accuracies.

8. Future Work

- Integration of text-to-speech: Integrating text-to-speech will make the application 100% stand-alone, giving a real-time translation of the ASL.
- Explore how environmental factors such as natural light, and blurry images, could affect the prediction accuracy.

9. Applications

The applications of such a system bridge the communication gap between people who use spoken language and those who the ASL.

10. Conclusion

After our exploratory analysis comparing the performance of various models, we successfully integrate OpenCV to develop a real-time American Sign Language interpreter for all 29 classes. The code for this application can be found HERE and the demonstration of this application can be found HERE.

11. Supplemental Items to Run Code

In order to run the notebook, we need to download the API authentication key (private to every user) from Kaggle and upload it to our notebook. The steps to do this are available HERE under the authentication section.

12. References

[1] Real-time American Sign Language Recognition with Convolutional Neural Networks, Research Paper by Brandon Garcia and Sigberto Alarcon Viesca, Stanford University.

[2] Deep Residual Learning for Image Recognition, Research Paper by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, IEEE Explore.

[3] Albelwi, Saleh, and Ausif Mahmood. 2017. "A Framework for Designing the Architectures of Deep Convolutional Neural Networks" *Entropy* 19, no. 6: 242.

[4] Residual Networks (ResNet). A tutorial on the ResNet architecture and its usage.

[5] Architecture of DenseNet-121. Article on the DenseNet architecture and it's usage.

[6] Densely Connected Convolutional Networks, Research Paper by Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger

[7] Ahmed KASAPBAŞI, Ahmed Eltayeb AHMED ELBUSHRA, Omar AL-HARDANEE, Arif YILMAZ, DeepASLR: A CNN based human computer interface for American Sign Language recognition for hearing-impaired individuals, Computer Methods and Programs in Biomedicine Update, Volume 2, 2022, 100048, ISSN 2666-9900, https://doi.org/10.1016/j.cmpbup.2021.100048

[8] Daroya, Rangel & Peralta, Daryl & Naval, Prospero. (2018). Alphabet Sign Language Image Classification Using Deep Learning. 0646-0650. 10.1109/TENCON.2018.8650241.

[9] (PDF) Alphabet Sign Language Image Classification Using Deep Learning Article on Real-Time ASL Classification.

[10] Codebase: Classification of the American Sign Language using Pytorch | by Helik Thacker | Medium

[11] Codebase: ASL Detection - 99% Accuracy | Kaggle

[12] Codebase: Benchmarking Model Training with a CPU | Kaggle