Image-Based American Sign Language Alphabet Translation using ResNet-18

Jerin Delbin Sebastian - (delbin.j@northeastern.edu)

Abstract

American Sign Language (ASL) is an important tool for hearing impaired people to use to communicate with others. There are around 500,000 ASL-users in the US but there is not enough ASL technology for people to use. The project aims to create an alphabet ASL translator by exploring ResNet-18. We use the dataset from Kaggle, which has 87000 training images across all 29 classes. The results we obtained from this tailor made network was a very impressive 97.40%.

Overview

American Sign Language (ASL) plays a critical role in enabling communication for individuals with hearing and speech impairments. It remains one of the most important and relevant standards of communication for that group of society. However, despite its importance, the integration of ASL into everyday life has been limited. Especially in today's day and age where everything is mostly online, standards like ASL are supposed to become useful, however their presence in these circles remain scarce. This project focuses on developing a deep learning model that can classify these ASL alphabet gestures thereby bridging this gap.

Since this is quite a vast and extensive dataset, preprocessing and EDA is a crucial step to essentially see what we are working with. The images in the dataset are of 200x200 pixel size, which can be computationally intensive, especially when working with deep neural networks. This fact fueled the decision to downsize it to a 32x32 size image to work with. This would speed up the training process greatly and make memory usage more efficient.

The approach that we intended here was to implement a neural network that has specifically been designed to work on image classification tasks. ResNet-18 is one such network that is pre-trained on the ImageNet dataset. Therefore, it already has enough knowledge about making decisions about features and boundaries. An additional argument for using ResNet-18 is to eliminate the **vanishing gradients** problem that is often faced by deep neural networks where the gradients get so small during backpropagation that the model doesn't learn effectively. ResNet counteracts this by employing **skip-connections**.

Therefore, all we are required to do is to change the last fully connected layer of the existing ResNet architecture to make it better suited to our 29 class application.

Initial process involved modifying the model to suit our application and training it on the preprocessed training dataset. We initially trained it on the whole training set that was given from Kaggle. I.e. It was trained on all 87,000 images without splitting into subsets. This ensured a very thorough and exhaustive training process. Post which we tested it on the Kaggle supplied testing set of 29 images**.** Although this seems to be an ideal result at first, it cannot be conclusive due to the limited number of images available in the testing set. Additionally, we also run the risk of the model being overfitted to the training data as well.

To counteract this potential issue, we implemented a second similar model but trained it differently. This time, we split the existing 87,000 images' training set into testing and validation subsets. This ensures that the model will train on fewer images and test and validate itself to a much larger number. This could potentially improve the model's ability to generalize to unseen data.

Architecture of ResNet-18

- **Input Layer:** The default ResNet-18 accepts 3-Channel RGB images. In our dataset, the images were grayscale, so the first convolutional layer was modified to accept single-channel input by changing the input dimension from 3 to 1.
- **Fully Connected Output Layer:** The original FC layer will output based on the ImageNet dataset it was trained on. We change it with a Linear Layer outputting for the 29 classes. This will extract the number of input features from the previous layer of the model and ensure it is suited for the 29 classes requirements.
- **Pre-trained Weights:** The rest of the model will retain its pre-trained weights. This is because these were the weights that were trained on the ImageNet dataset and is what we will be working off of.

The deep structure of ResNet-18 when combined with these modifications result in a model that is excellently able to distinguish between features within the images in the dataset.

Experimental Setup

1. **Dataset:** We used a dataset from Kaggle, which consists of 29 folders corresponding to one of 26 English Alphabets and 3 special characters (space, delete and nothing) with each folder having 3000 images each totalling 87000 training images. Additionally, the testing folder has 29 testing images across all 29 classes. Upon looking through the pictures manually, we understand that the data is very clean and balanced.
2. **Computational Environment:** We used a jupyter notebook to run the project. We ran a jupyter notebook Here is the version information of the libraries and hardware we used in this project.

- Python 3
- Pandas 2.2.2
- Numpy 1.26.4
- Cv2 4.10.0
- Matplotlib 3.8.4
- Seaborn 0.13.2
- Sklearn 1.4.2
- Tensorflow '2.18.0
- Pytorch 2.5.1
- Keras 3.7.0
- CPU: Intel Core i7
- RAM: 16 GB

3. **Data Preprocessing:** The original size of the image is 200 x 200, which is too large and takes too long to train especially with a deep network like in this case. This was the rationale behind reducing the image size to a 32x32 size. The image was also reduced to a 1-D array during preprocessing. However, since ResNet takes 3-channel RGB as input, we were required to resize it.

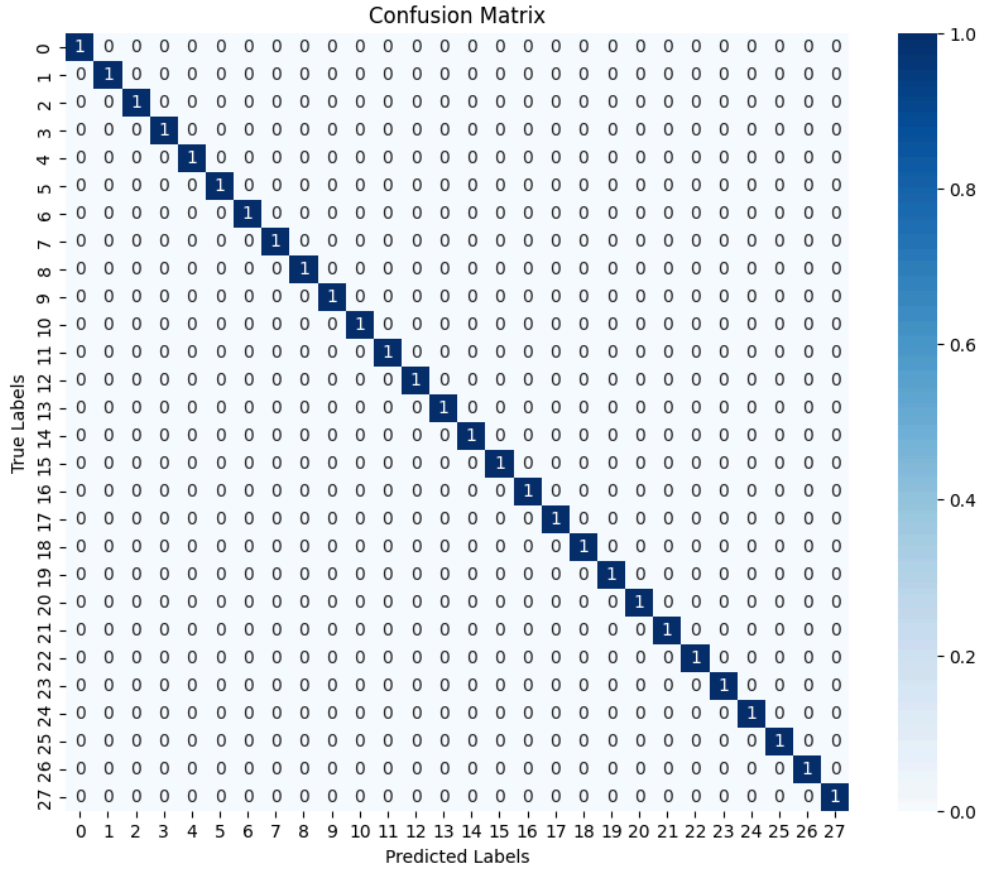
4. **Model Implementation:**

   ResNet-18:

   - Implemented from scratch using Pytorch
   - Dataset is pretrained on ImageNet
   - 10 epochs.
   - The input images were passed through the modified ResNet network to generate class logits for 29 classes
   - Loss was computed using **Cross-Entropy** Loss between predicted logits and ground-truth labels
   - Adam optimizer was used to update model parameters and minimize loss.

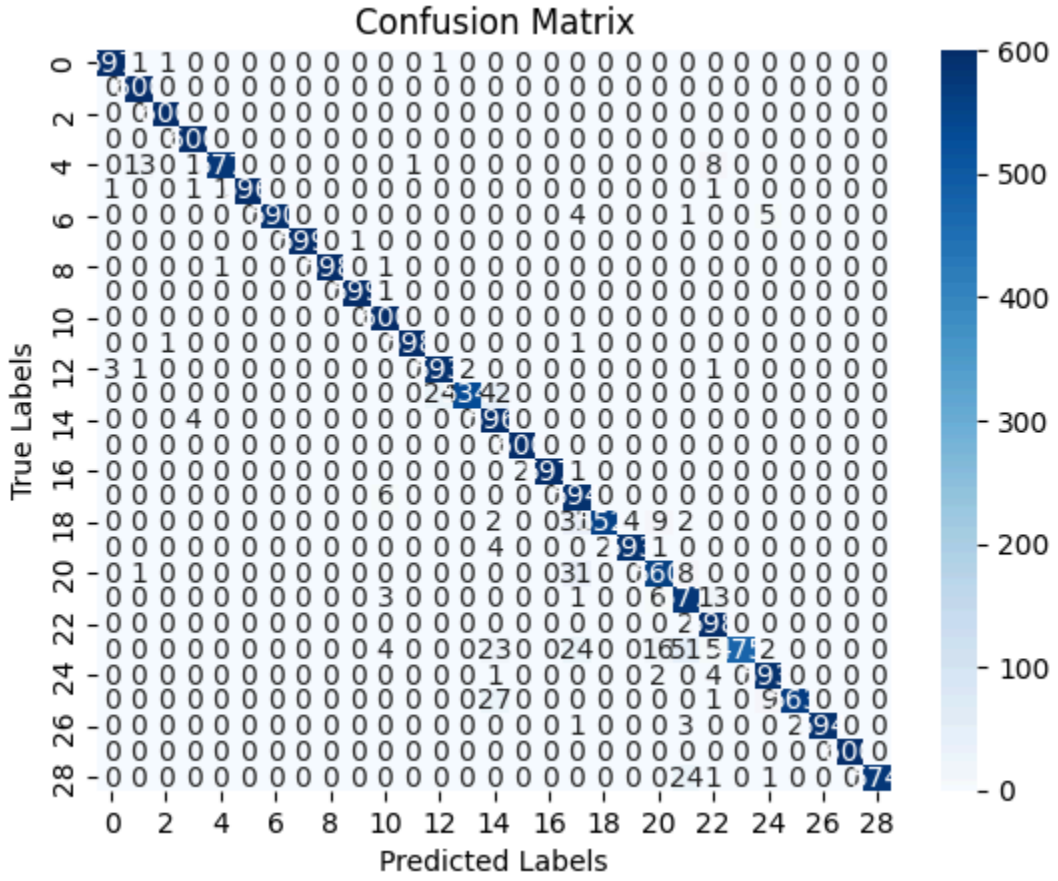
Experiment results:

The modified custom ResNet-18 reached a high 98.2% accuracy on the training data and 100% accuracy on the validation data when using athe complete dataset.



Confusion Matrix that yielded 100% testing accuracy when trained on 87,000 images and testing on 29 images.

Additionally, we obtained a 97.4% accuracy when split into training, validation and testing data. These results show that the model learns well and performs strongly on unseen data.



Confusion Matrix that yielded 97.40% testing accuracy when trained on train-test-split conditions

Conclusion

This project showed how ResNet-18 can be used for recognizing ASL gestures by adjusting the model to work with grayscale ASL images. The model achieved a **97.4%** accuracy on a dataset split into training, testing and validation sets, and it scored **100%** accuracy on a Kaggle test set. These results highlight how well ResNet-18 works for classifying ASL gestures with high accuracy and reliability.

ResNet-18 is a more advanced model, made stronger by its "residual connections" that help it identify complex patterns. While it requires more computing power, it proves to be a good fit for tasks needing accurate ASL recognition. Future improvements could focus on making the model

faster for real-time use or expanding it to recognize dynamic ASL gestures, making it even more useful in accessibility and communication tools.

References

- https://www.geeksforgeeks.org/resnet18-from-scratch-using-pytorch/
- https://debuggercafe.com/implementing-resnet18-in-pytorch-from-scratch/
- https://www.kaggle.com/code/ivankunyankin/resnet18-from-scratch-using-pytorch
- https://www.kaggle.com/datasets/grassknoted/asl-alphabet