Image-Based American Sign Language Alphabet Translation

Ngoc Khanh Vy Le

Abstract

American Sign Language (ASL) is an important tool for deaf 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 translation by exploring different approaches and comparing them to find the best performer - Logistic Regression, Convolutional Neural Networks (CNN) and ResNet-18. We use the dataset from Kaggle, which has 87000 training images across all 29 classes. The binary logistic regression is extremely efficient with high accuracy, but the multiclass logistic regression model does not perform well compared to others - accuracy is at 49%. CNN model with 3 hidden layers - 32, 64, 64 Filters and 32 x 32 image size - has the best performance, 97%. Our ResNet-18 model also performs relatively well at 94%.

Overview

These days, online interactions are extremely crucial. A lot of people prefer to commute online instead of having in-person conversation. In addition, it is also becoming challenging for ASL-users to commute through online, especially meeting new friends or doing interviews since there are not enough available tools for them to use and new technology is often expensive. In our project we aim to develop an ASL tool that can translate ASL alphabet language using computer vision.

After carelly exploring the dataset, we decided to reduce our image size to 32 x 32 to increase the training speed, memory efficiency and reduce noise. Furthermore, we tried different approaches, implemented different models, different optimization using the same preprocessed dataset to find the best performing model. In this project, we implemented binary logistic regression for two similar and two different letter signs, as well as a multiclass binary logistic regression across all 29 classes. We chose the Logistic Regression model as a baseline model because it is a common, efficient and simple model. Our second model is ResNet-18, we implemented it from scratch using Pytorch and ran it with 3 epochs. Also, we implemented six different CNN models - each of them has different hidden layers, different filters run in 10 epochs with batch size of 64. We used different metrics such as accuracy, f1-score, precision, recall, confusion matrix for model performance comparasion.

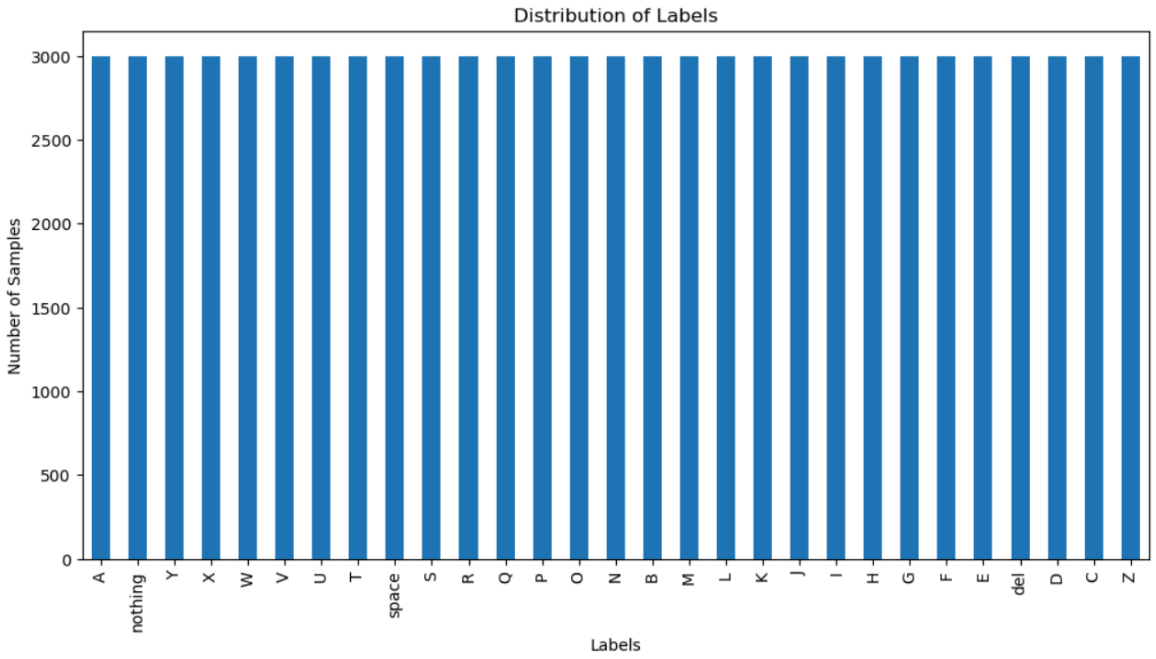
Both of our binary logistic regression models perform extremely well, at 90% accuracy for the model classifying two similar letter signs and 100% for the model classifying two different letter signs. However, the multiclass logistic regression does not perform well as we expected, 49% accuracy. Our fifth CNN model (3 hidden layers - 32, 64, 64) performs the best

among others models at 97% accuracy. Our ResNet-18 model performs well at 94% of accuracy but we were expecting it should be the best performing model overall.

We choose these models because both CNNs and ResNet-18 are known for their accuracy and are highly effective for image recognition and classification tasks. Additionally, the primary reason we can think why the ResNet-18 does not outperform the CNN model is due to the dataset characteristics. The ResNet-18 is designed for large and complex datasets with high-resolution images but our dataset is considered as small for computer vision model implementation tasks and we also reduced the image size to 32 x 32 for better training speed and memory efficiency.

Experimental Setup

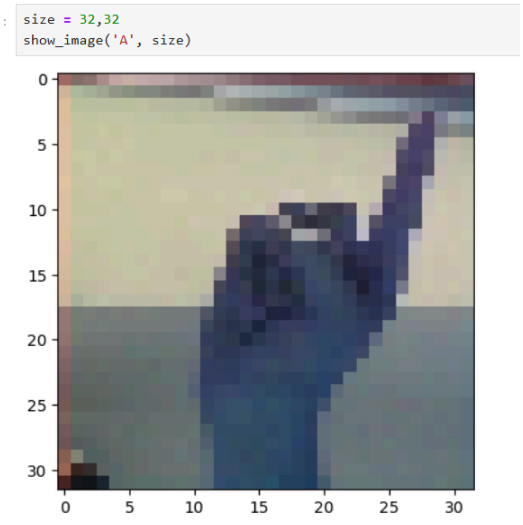
1. **Dataset:** We use a dataset from Kaggle, which consists of 87000 training images, 29 testing images across all 29 classes. The data is extremely well-documented and clean and we did not have to perform any cleaning process. The dataset is balanced, all the labels have the same occurrences.

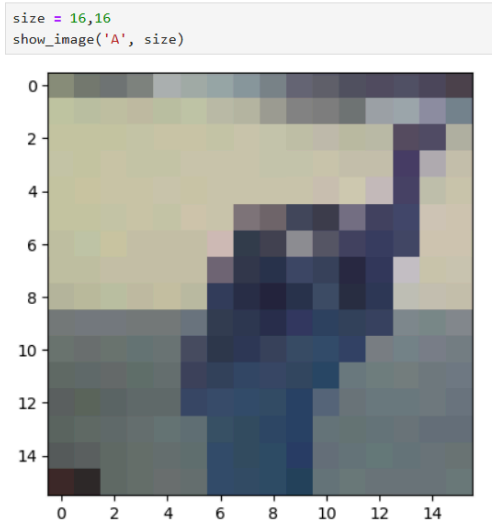


2. **Computational Environment:** We used a jupyter notebook to run the project (if you want to run it in the google colab or other software, make sure to change the data path when exporting the dataset). We ran a jupyter notebook through anaconda and our anaconda is the latest version at the time we did the project. 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
- GPU: NVIDIA GTX 3070 Ti
- CPU: Intel Core i7
- RAM: 16 GB

3. **How to run the project:** sometimes the code may not work if the library versions are mismatched (but we think it will be fine in this project).
    - After exporting the dataset, the images are stored in nested directories with double paths: “asl_alphabet_train\\asl_alphabet_train\\...”, make sure to move the image/image files to a single-level directory.
    - The whole file takes around 3 hours to run in my setup
4. **Experiential Preprocessing:** The original size of the image is 200 x 200, which is too large and takes too long to train.. We ran our input images in different sizes in some simple models to monitor the training speed. We chose 32 x 32 as a image size to train all the models and 16 x 16 for alternative reduced size to run in some other cases.





5. **Models:**
    a. **Logistic Regression:** The model acts as a baseline model for us to have better ideas of how our other models will perform. We ran two different logistic regression models - binary and multiclass because we wanted to know how significantly different the performances of these models are.
        - Binary logistic regression: classifying two similar and two different letter signs
        - Multiclass logistic regression: classifying all across 29 labels.

b. **CNNs:** Six CNN architectures with varying numbers of hidden layers and filters
   - 64 batch size
   - 10 epochs
   - Grey-scale 32 x32 and 16 x 16 images
   - Run in both 2 different image sizes
   - Convolutional layers: Various configurations 16, 32, 64 layers
   - Activation: ReLU after each convolutional layer.
   - Pooling: Max pooling after each convolutional block.
   - Fully connected layers: Dense layers, a softmax output layer

c. **ResNet-18:**
   - Implemented from scratch using Pytorch
   - Dataset is pretrained on ImageNet
   - 3 epochs.

Experiment results

1. **Logistic Regression:**

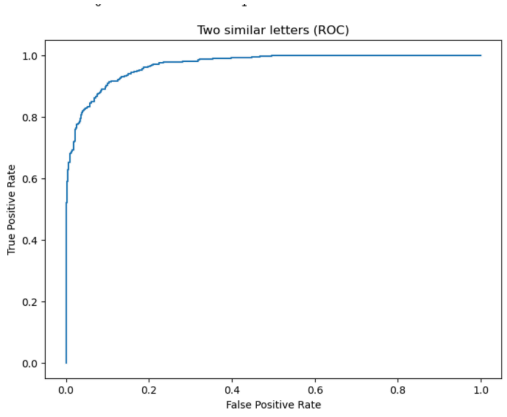
a. **Binary Logistic Regression - Two similar letter signs:** the model performs well with the accuracy at 90%. Other metrics also show the high score, 90% for all . The AUC Score at 96% indicates that the model has a high ability to distinguish between the classes.

```
Accuracy: 0.8991666666666667
Recall: 0.8989928605373079
Precision: 0.8992491657397108
F1_score: 0.8990903537466877
Classification Report:
              precision    recall  f1-score   support

           A       0.90      0.91      0.90       613
           E       0.90      0.89      0.90       587

    accuracy                           0.90      1200
   macro avg       0.90      0.90      0.90      1200
weighted avg       0.90      0.90      0.90      1200

AUC Score: 0.9694023027476787
```



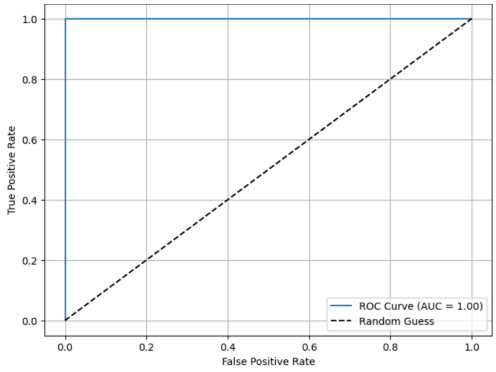
b. **Binary Logistic Regression - Two different letter signs:** the model performs extremely well, with 100% in all the metrics, showing that the model can completely distinguish between 2 classes that are completely different.

```
Accuracy: 1.0
Recall: 1.0
Precision: 1.0
F1_score: 1.0
Classification Report:
              precision    recall  f1-score   support

           C       1.00      1.00      1.00       613
           G       1.00      1.00      1.00       587

    accuracy                           1.00      1200
   macro avg       1.00      1.00      1.00      1200
weighted avg       1.00      1.00      1.00      1200

AUC Score: 1.0
```



c. **Multiclass Logistic Regression:** although the binary models perform well, the multiclass model has the weakest performance among the other models - 49% across all other metrics. The metrics for each class are also different. The class like *nothing* performs extremely well ~90%, while some of the classes like *O, U, V*, and *space* perform poorly.

```
Accuracy: 0.49333333333333335
Recall: 0.49436993011483543
Precision: 0.4966779249466738
F1_score: 0.49349252144411004
Classification Report:
              precision    recall  f1-score   support

           A       0.53      0.51      0.52       584
           B       0.49      0.48      0.48       574
           C       0.55      0.58      0.57       617
           D       0.46      0.44      0.45       589
           E       0.49      0.43      0.46       580
           F       0.53      0.45      0.49       585
           G       0.52      0.51      0.51       587
           H       0.55      0.56      0.56       599
           I       0.52      0.58      0.55       598
           J       0.57      0.49      0.53       610
           K       0.54      0.58      0.56       609
           L       0.65      0.61      0.63       641
           M       0.39      0.39      0.39       626
           N       0.51      0.51      0.51       587
           O       0.39      0.42      0.40       569
           P       0.62      0.64      0.63       603
           Q       0.59      0.65      0.61       586
           R       0.46      0.35      0.40       649
           S       0.50      0.46      0.48       571
           T       0.39      0.37      0.38       660
           U       0.39      0.32      0.35       619
           V       0.32      0.42      0.36       568
           W       0.45      0.31      0.36       580
           X       0.42      0.49      0.45       596
           Y       0.40      0.40      0.40       584
           Z       0.40      0.45      0.42       579
         del       0.51      0.62      0.56       602
     nothing       0.90      0.88      0.89       568
       space       0.36      0.43      0.39       680

    accuracy                           0.49     17400
   macro avg       0.50      0.49      0.49     17400
weighted avg       0.50      0.49      0.49     17400
```

Overall, the Logistic Regression model performs significantly better than we expected.

2. **CNNs:**

We ran with six different CNN models and our fifth model is the one that is the best performer. In this model, we ran with 3 hidden layers (32, 64, 62) and image size at 32 x 32 pixels. We add drop out to avoid overfitting although it may not be necessary in this case since the dataset is already balanced.

Model: "sequential_4"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 30, 30, 32)	320
max_pooling2d_6 (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_10 (Dropout)	(None, 15, 15, 32)	0
conv2d_7 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_7 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_11 (Dropout)	(None, 6, 6, 64)	0
conv2d_8 (Conv2D)	(None, 4, 4, 64)	36,928
max_pooling2d_8 (MaxPooling2D)	(None, 2, 2, 64)	0
dropout_12 (Dropout)	(None, 2, 2, 64)	0
flatten_4 (Flatten)	(None, 256)	0
dense_8 (Dense)	(None, 128)	32,896
dropout_13 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 29)	3,741

Total params: 92,381 (360.86 KB)
Trainable params: 92,381 (360.86 KB)
Non-trainable params: 0 (0.00 B)

We ran 10 epochs, the val_loss significantly decreased each epoch showing no overfitting. In addition, the results of this CNN model is also the highest among all the models, scoring 97% in all the metrics - accuracy, f1-score, precision, recall.

```
cnn_models(model3_32, target_size)
```

```
Epoch 1/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 15s 12ms/step - accuracy: 0.1794 - loss: 2.8261 - val_accuracy: 0.6714 - val_loss: 1.1309
Epoch 2/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - accuracy: 0.5656 - loss: 1.3225 - val_accuracy: 0.8021 - val_loss: 0.6511
Epoch 3/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - accuracy: 0.6854 - loss: 0.9346 - val_accuracy: 0.8796 - val_loss: 0.4341
Epoch 4/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - accuracy: 0.7360 - loss: 0.7677 - val_accuracy: 0.8997 - val_loss: 0.3381
Epoch 5/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - accuracy: 0.7809 - loss: 0.6468 - val_accuracy: 0.9241 - val_loss: 0.2623
Epoch 6/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - accuracy: 0.7996 - loss: 0.5773 - val_accuracy: 0.9299 - val_loss: 0.2497
Epoch 7/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 12s 11ms/step - accuracy: 0.8182 - loss: 0.5272 - val_accuracy: 0.9501 - val_loss: 0.1927
Epoch 8/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - accuracy: 0.8346 - loss: 0.4761 - val_accuracy: 0.9568 - val_loss: 0.1690
Epoch 9/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - accuracy: 0.8421 - loss: 0.4619 - val_accuracy: 0.9633 - val_loss: 0.1525
Epoch 10/10
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 13s 12ms/step - accuracy: 0.8545 - loss: 0.4208 - val_accuracy: 0.9666 - val_loss: 0.1371
```

3. **ResNet-18:**

   The ResNet-18 model showed good performance, with 94% accuracy. We ran 3 epochs and each epoch takes around 21 minutes to run. The loss during each epoch is small and also significantly decreased each epoch.

```
Epoch 1/3: 100%|██████████| 2719/2719 [21:44<00:00,  2.08it/s, Loss=0.174]
Epoch 1 Loss: 0.4119851107501198, Accuracy: 0.8651724137931035
Epoch 2/3: 100%|██████████| 2719/2719 [21:20<00:00,  2.12it/s, Loss=0.0782]
Epoch 2 Loss: 0.06990886174994111, Accuracy: 0.9775977011494252
Epoch 3/3: 100%|██████████| 2719/2719 [21:19<00:00,  2.12it/s, Loss=0.0257]
Epoch 3 Loss: 0.0441159194669313, Accuracy: 0.9865747126436781
Final accuracy: 0.9431149425287356
```

Conclusion

The results show that our model performs extremely well, metrics are high and no overfitting. The Logistic regression models perform well, achieving reasonable accuracy in binary classification tasks but struggling with multi-class classification. The performance is better than I expected for a baseline model. CNNs showed significant improvements. The ResNet-18 also performs well but does not perform better than the CNN.

Discussion

The logistic regression performs poorly when running more than two classes due to its limitation in handling complex multiclass problems. However the binary model does extremely well, so we can consider this logistic regression model for image classification with less than two classes.

As expected in both CNN and ResNet-18 which are known for their performance in handling multiclass image classification tasks, their performances are extremely well >95% of accuracy and in other metrics. However, we were expecting the ResNet to perform better than CNN, this could be due to the dataset characteristics and number of epochs we ran. Increasing

the number of epochs could make the performance better. We can also change the number of hidden layer filters in CNN models to find the better performance.

In addition, we noticed that the performance will decrease when we reduce the image size to 16 x 16. The current models are already showing the best performance but if we increase the size of the image we can achieve better model performance.

References

- https://www.tensorflow.org/tutorials/images/cnn
- 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