Fingerspelling Interpretation

Arunima Ayshee
160204066@aust.edu

Tahira Salwa Rabbi Nishat
160204070@aust.edu
Zannatul Ferdous
13.02.04.070@aust.edu

1. Introduction

Fingerspelling is a subset of sign language used for communication by representing a word or other expression by rendering its written form letter by letter in a manual alphabet. It is a method of spelling words using hand movements.

In this project, we have worked with 26 different letters and 3 different symbols of daily use of the English language. We are making a system that can recognize hand symbols and give us the corresponding meaning as output. A graphic illustration of American Sign Language is shown in below figure

:

Fig 01: American Sign Language

a. **Motivation:**

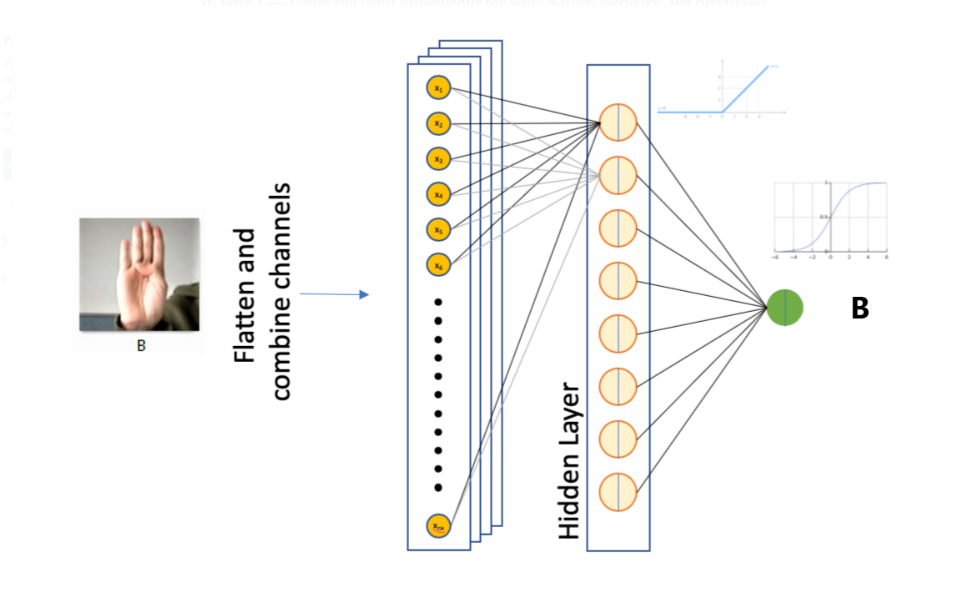
a) Fingerspelling is a survival tool for deaf people when it comes to learning, communicating with rest of the world. It bridges the gap between the learner's language and sign language. Some deaf people like to use fingerspelling more than others. When we are naming something like a book, people, movie, brands, and places it is easier to fingerspell because there is not always a symbol for the proper name we are trying to fingerspell. For instance, we cannot say "K- Mart" without spelling it out. The idea of fingerspelling is to give us another method in sign language to help the betterment of understanding all around, especially with names that don't really have signs.

b) Some people we come to meet did not grow up learning sign language, instead used a coding system like SEE or the Rochester method (full fingerspelling of English sentences). Also, the community of deaf people comes from a variety of backgrounds, and each background has varying levels of fingerspelling. That's why it's an important skill to master so that we can understand deaf individuals when we come in contact with them. Moreover, it also allows us to learn from deaf people and helps to understand the unknown meaning of signs used by them during a conversation.

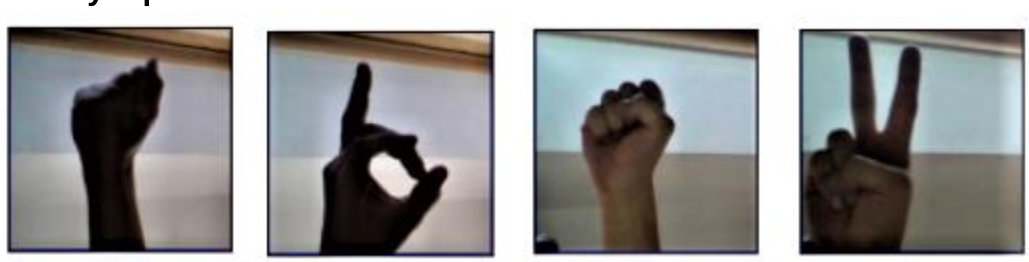
c) We had to work with RGB images with high resolution. So we had to convert it into grayscale and Transform them into what we can feed to network . Also, the length of the dataset was huge. So the code took a long time to run.

2. **Related Works**
   - **a.** Object detection, detecting ASL using webcam, real time ASL testing.
   - **b.** Most of the works have been done on ASL till now is based on Convolutional Neural Network, other real time ASL detection work, they have used YOLO which is an object detection neural network based on the CSP approach, Some work were done using Resnet.
   - **c.** We have used shallow Neural Network for our project, it simplified the task maintaining a good accuracy.

3. **Project Objective**(s)
   a. Tasks of the system
      i. Data Preprocessing: We have taken our dataset from Kaggle ASL dataset.
         We had to add label for each class of data. Then we calculated how many images of each class we have in our training data. All of the classes have got equal training data.
      ii. We processed the dataset, transform the images into 224*224 sized grayscale images using torchvision.transforms .
      iii. Then we made our dataset using Imagefolder which returned the images and labels.
      iv. We split the dataset into train:validation(80:20) set.
      v. We used Dataloader using this dataset.
      vi. Then, we created our model NeuralNetworkModel(), trained it with train data set, and tested it with the validation set.



      vii. Finally we tested our model and evaluated its performance.
   b. dummy input:



dummy output:

4. **Methodologies / Model**
    a. We have discussed this part in section 3

5. **Experiments**
    a. **Dataset**
        i. Our Dataset is ASL dataset from Kaggle:
        [kaggle datasets download -d grassknoted/asl-alphabet]

        The total number of images in our dataset is 87,000.

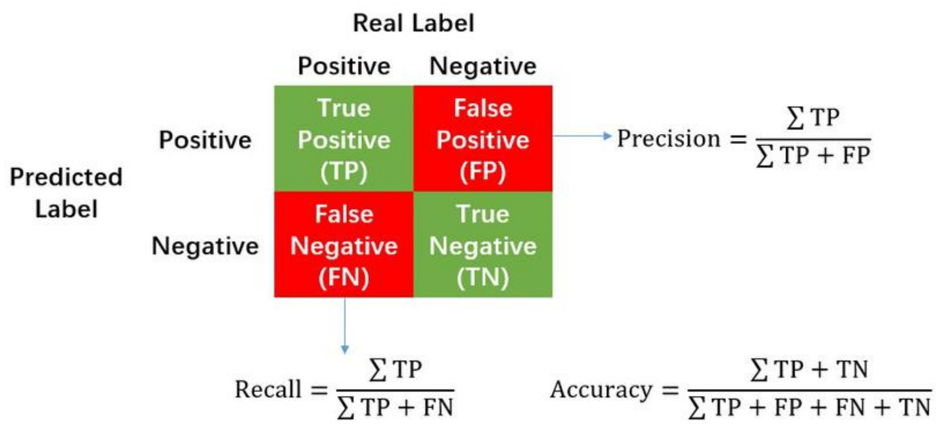
        We Have 29 classes from A-Z, and del, nothing, space. Each class contains 3,000 images.

        ii. Here is a sample of our dataset where we have showed one first picture from each classes.

iii. Our total dataset from Kaggle had two directory, train directory and test directory. We split the train data from train directory into train: validation = 80:20, so the length of train dataset was 69600 and validation dataset was 17400. The Test directory from Kaggle 28 test images, but we need at least 29 images, one from each classes to evaluate our model. So, we made our own test directory, it had 29 images one from each class.

b. Evaluation Metric

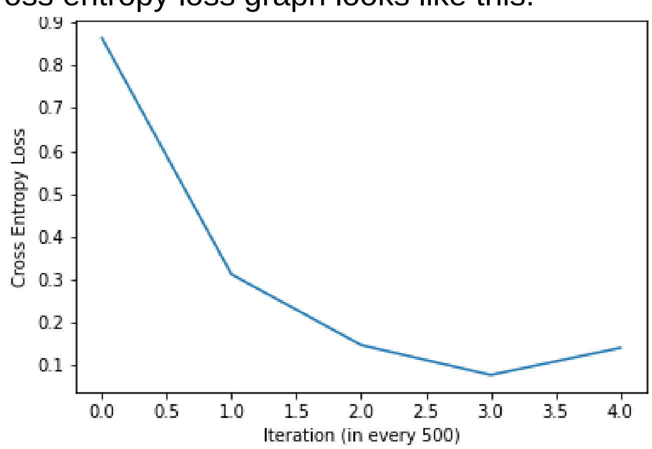
i. We evaluated our model on the basis of accuracy. Then we made the evaluation matrix and calculated precision, recall, f1-score done on validation set. We have measured the accuracy for our test dataset too.



c. Results

i. Here are our best 3 results we have gotten so far:

1. We have achieved 97.29885057471265% accuracy and the cross entropy loss graph looks like this:

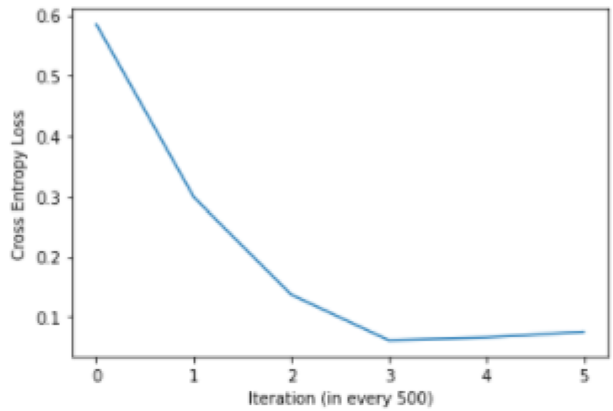


It looks a little bit overfitted. But all of our test data were classified correctly.

The evaluation matrix looked good too.

	precision	recall	f1-score	support
0	0.88	0.89	0.89	2815
1	0.89	0.92	0.91	3130
2	0.95	0.96	0.95	3075
3	0.94	0.92	0.93	2845
4	0.92	0.86	0.89	2955
5	0.94	0.96	0.95	2870
6	0.93	0.95	0.94	2870
7	0.97	0.96	0.96	3035
8	0.92	0.87	0.90	3250
9	0.90	0.96	0.93	3150
10	0.89	0.92	0.90	2860
11	0.94	0.94	0.94	2895
12	0.92	0.84	0.88	3150
13	0.96	0.98	0.97	2855
14	0.80	0.90	0.85	2940
15	0.94	0.98	0.96	2985
16	0.98	0.94	0.96	2935
17	0.85	0.79	0.82	3095
18	0.90	0.83	0.86	3090
19	0.88	0.86	0.87	3175
20	0.85	0.75	0.80	2960
21	0.88	0.78	0.83	3135
22	0.79	0.82	0.80	2855
23	0.91	0.88	0.89	2985
24	0.85	0.91	0.88	2935
25	0.92	0.90	0.91	3045
26	0.88	0.94	0.91	3040
27	1.00	1.00	1.00	3210
28	0.75	0.92	0.82	2860
accuracy			0.90	87000
macro avg	0.90	0.90	0.90	87000
weighted avg	0.90	0.90	0.90	87000

2. We have achieved 98.79885057471265% accuracy and the cross entropy loss graph looks like this:



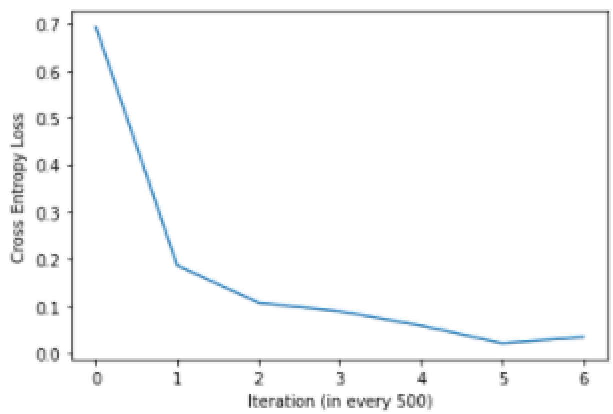
It looked better. And all of our test data were classified correctly.

```
Right: 29  Wrong:  0
right Assumed: 100.0 % Wrong Assumed: 0.0 %
```

The evaluation matrix looked better too.

	precision	recall	f1-score	support
0	0.93	0.93	0.93	3378
1	0.96	0.93	0.94	3756
2	0.97	0.96	0.97	3690
3	0.94	0.97	0.95	3414
4	0.93	0.92	0.92	3546
5	0.97	0.96	0.97	3444
6	0.97	0.95	0.96	3444
7	0.97	0.97	0.97	3642
8	0.93	0.93	0.93	3900
9	0.97	0.96	0.96	3780
10	0.93	0.93	0.93	3432
11	0.94	0.97	0.95	3474
12	0.98	0.90	0.93	3780
13	0.99	0.98	0.98	3426
14	0.91	0.95	0.93	3528
15	0.99	0.97	0.98	3582
16	0.98	0.96	0.97	3522
17	0.91	0.83	0.87	3714
18	0.86	0.92	0.89	3708
19	0.93	0.93	0.93	3810
20	0.86	0.85	0.86	3552
21	0.91	0.84	0.87	3762
22	0.78	0.90	0.83	3426
23	0.93	0.94	0.93	3582
24	0.90	0.93	0.92	3522
25	0.96	0.95	0.95	3654
26	0.94	0.94	0.94	3648
27	0.99	1.00	1.00	3852
28	0.92	0.96	0.94	3432
accuracy			0.94	104400
macro avg	0.94	0.94	0.94	104400
weighted avg	0.94	0.94	0.94	104400

3. We have achieved 98.95402298850574% accuracy and the cross entropy loss graph looks like this:



It looked nice. And all of our test data were classified correctly.

```
Right: 29  Wrong:  0
right Assumed: 100.0 % Wrong Assumed: 0.0 %
```

This is how our model evaluation was for this case.

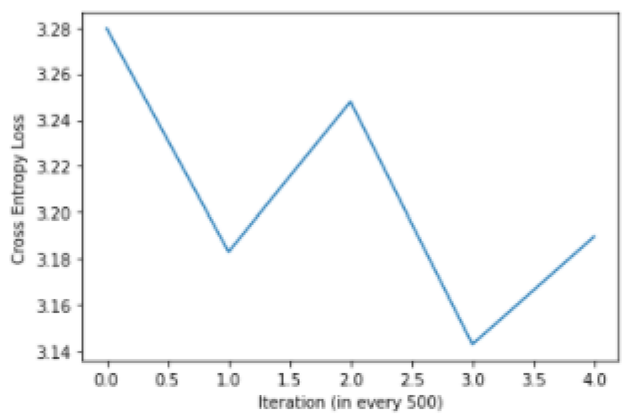
0	0.94	0.95	0.94	3941
1	0.95	0.95	0.95	4382
2	0.99	0.96	0.98	4305
3	0.94	0.97	0.96	3983
4	0.94	0.93	0.94	4137
5	0.97	0.97	0.97	4018
6	0.96	0.98	0.97	4018
7	0.97	0.97	0.97	4249
8	0.94	0.94	0.94	4550
9	0.98	0.96	0.97	4410
10	0.93	0.93	0.93	4004
11	0.98	0.96	0.97	4053
12	0.94	0.94	0.94	4410
13	0.99	0.99	0.99	3997
14	0.94	0.95	0.95	4116
15	0.99	0.98	0.99	4179
16	0.98	0.98	0.98	4109
17	0.93	0.89	0.91	4333
18	0.90	0.96	0.93	4326
19	0.91	0.95	0.93	4445
20	0.90	0.86	0.88	4144
21	0.86	0.89	0.88	4389
22	0.95	0.87	0.90	3997
23	0.95	0.96	0.95	4179
24	0.92	0.96	0.94	4109
25	0.95	0.97	0.96	4263
26	0.96	0.96	0.96	4256
27	1.00	1.00	1.00	4494
28	0.98	0.92	0.95	4004
accuracy			0.95	121800
macro avg	0.95	0.95	0.95	121800
weighted avg	0.95	0.95	0.95	121800

No.	batch_size	num_iters	learning rate	optimizer	Accuracy
1	120	3000	0.01	Adagrad	97.29885057471265%
2	120	4000	0.01	Adagrad	98.79885057471265%
3	160	4000	0.01	Adagrad	98.95402298850574%

ii. Our dataset was big. So, we took batch size between 100-160. For learning rate we took either 0.001 or 0.01, but 0.01 gave better result. Hidden layer was 200.
First, we tried with deep neural network Linear->Tanh->Linear->LeakuRelu-> Linear->Relu6->Linear->Sigmoid with and Adamax optimizer and cross-entropy loss function. The result was poor. Nearly 30%.

```
Iteration: 500. Loss: 3.2796342372894287. Accuracy: 17.632183908045977
Iteration: 1000. Loss: 3.1829161643981934. Accuracy: 20.971264367816094
Iteration: 1500. Loss: 3.2478904724121094. Accuracy: 22.724137931034484
Iteration: 2000. Loss: 3.1428582668304443. Accuracy: 24.201149425287355
Iteration: 2500. Loss: 3.189634084701538. Accuracy: 25.080459770114942
```

The cross-entropy loss graph was bad too.



We used this combination in our model with various hyperparameter but result didn’t change significantly.

So, we changed the model and used four layer Linear network for fc1->fc2->fc3->fc4 while varying hidden layer dimension for each layer 224*224->512->256->128->29 and forwarded them using ReLU. The accuracy spiked over 90%. We later tuned more and got 98% accuracy.

6. Conclusion

This system is designed specifically for deaf and hard of hearing people/students. With the help of the fingerspelling method, one can spell the letters of a word in order to get to the meaning. When a person learns sign language, the first thing he learns is the alphabet, the alphabet is going to be signed with the fingers. And once he had learned the alphabet then he will move on to fingerspelling. But a hard situation may arise if they come to appear before a person who can speak and/or without any knowledge of fingerspelling. Our system wants to help those people and make it easier by letting them express their words to the world.