CV for American Sign Language

Hardik Prajapati
hprajapa@usc.edu

Meghana Achanta
vachanta@usc.edu

***Abstract*— We propose to design a Deep Learning Model to read the American Sign Language using PyTorch. Our project is a multi-class image classification model which will identify the 29 classes: 26 alphabets of the English language and three additional characters: SPACE, DELETE and NOTHING. Along with this, we aim to implement live-video classification such that our project can recognize the gestures in real-time.**

Keywords—Computer Vision, Convolutional Neural Networks (CNN), American Sign Language (ASL), Fully Connected Layer

Github Link: **Github Repository**

I. Introduction

Sign language is an important communication medium for hearing-impaired people. One in eight people in the United States (13%, or ~30 million) aged 12 years or older has hearing loss in both ears. About 2 percent of adults aged 45 to 54 have disabling hearing loss. Sign language is a language of hand gestures, which on interpretation gives some meaning. Across the world, every country has their own sign language. For this project we are concentrating on the recognition of American Sign Language (ASL). ASL is a standardized sign language, however only 250,000-500,000 understand it [1], which makes it difficult for the users dependent on ASL to converse in real-life scenarios. We focus on the recognition of the alphabet and 3 other signs which would be the core of the language.

II. Literaure review

Thad Starner and Alex Pentland [2] proposed Hidden Markov Model for the real-time American Sign Language using desk and wearable computer-based video. The designed model was used to recognize personal pronouns, certain verbs, nouns, adjectives and not to identify the entire lexicon of the ASL. The tracking system was used not to attempt a fine description of the hand gesture but was rather aimed to concentrate on the evolution of gestures over time. The system was used efficiently and obtained an accuracy of 94%. The major limitation of HMM is that it has context dependency.

Izzah and Suciati [3], used Generic Feature Descriptor (GFD) for feature extraction and K-Nearest Neighbors(K-NN) to classify the signs. Their system was able to recognize 120 stored images in a database and 120 images which were captured real-time using a webcam. Their system had an accuracy of 86% for the stored images and 69% for the images captured using webcam. This paper provides an overview of the classical ML techniques used to classify the ASL characters. Bajaj and Malhotra [4], had used K-Nearest Neighbors, Random Forest Classifier and Neural networks separately to recognize characters of ASL. For classifying using K-NN of a record t, the K-Nearest Neighbors is retrieved and they form a neighborhood of t [15]. This largely depended on the value of k they used and hence they used values of k in the range (1,25). They obtained an accuracy of about 92.71% for this method. While implementing random forests, to find the optimal value of the number of decision trees, they tested it for values of n in range (1,200). The random forest classifier obtained an accuracy of 93.4%. Lastly, they modeled a neural network in Keras with 9 dense layers, where the initial 8 layers were powered by ReLU activations, and the final layer used SoftMax as an activation. They had used Categorical Cross Entropy Loss with Adam optimizer and trained the model for 128 epochs to obtain a final accuracy of 95.6%.

Our model will use Convolutional Neural Network and MLP to classify the ASL characters and observe the differences between the accuracies, confusion matrixes generated, and the training time taken by the model.

The report has been divided into seven sections: The first section discusses about the problem, the second section gives an overview about the existing methods used, the third section describes the model architectures that we will be implementing, the fourth section provides the methodology used, the fifth section provides a description of the performance results obtained for the different experiments and the sixth section is the conclusion with future scope. The seventh section concludes the report with the references.

III. Model Architecture

For our project, we have implemented two models. The first model is the adaption of LeNET architecture, the second one is a Multi-Layer Perceptron i.e., using just the fully connected layers. We tried four settings with these architectures.

A. LeNET Architecture:

We implemented LeNET-2 Architecture using two convolution layers using maxpooling and ReLU activations with multi-cross entropy loss. In the first convolution layer, we have rescaled the image to 32x32. The total number of input activations for the first layer is 3072, the total number of output activations being 1176, the filter coefficients being 450 and the total number of trainable parameters in this conv2D being 450.

For the second convolution layer, the input size being 14x14, with 6 input channels and 16 output channels and filter size of 5x5. The total number of input activations is 1176, the total number of output activations being 400. The total number of filter coefficients is 2400 and the total number of trainable parameters is also 2400.
The first hidden layer has 400 weights, the second hidden layer has 120 weights, and the third layer has 84 weights

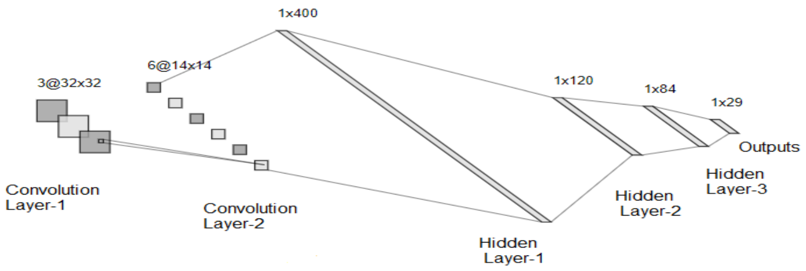
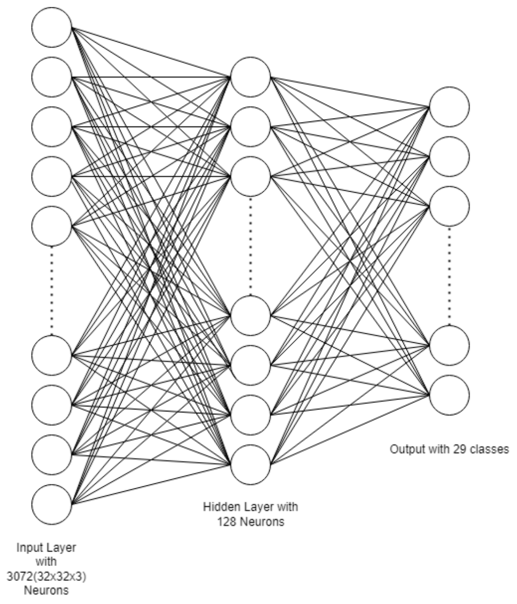


Figure 1. Model Architecture of LeNET[5]


Output with 29 classes
Hidden Layer with 128 Neurons
Input Layer with 3072(32x32x3) Neurons
Figure 2. Model Architecture of Multi-Layer Perceptron

which finally predicts 29 outputs. The model architecture has been depicted in Figure 1.

B. *Multi-Layer Perceptron*

In this model (Figure 2), we considered taking one hidden layer, with ReLU activation to predict the result. The first layer has about 128 weights, which is classified into 29 classes using SoftMax.

IV. Methodology

A. *Dataset Description:*

The dataset consists of labeled images of alphabets from American Sign Language. There are 29 classes which include alphabets A-Z, SPACE, DELETE & NOTHING.

*Number of data points: Training = 87000 images, Testing = 29 images

*Number of features or input variables: 40000(200x200) dim feature vector.

*Feature or input-variable types: Real-Valued pixel intensity

*Label (output) type: String (class name)

*Balanced set? Yes.

*Missing Data? No

*Any other comments on the dataset: None.

B. *Experiments:*

The problem at hand is supervised learning with multi-class classification task. To solve this problem, we have performed four experiments using Multi-Layer Perceptron and LeNET architecture. The models that we have designed are simple networks with a model size of 250kB (including all the trainable parameters). The dataset has been divided into a training set of about 87,000 images and a testing set of 29 images. We have obtained an accuracy of about 97% over the training set when the model is trained for 5 epochs. We have incorporated TensorBoard for visualizing the learning curves and accuracy which helps the user to keep track of the data. Adding to this, we have also performed real-time ASL classification to observe how well our model is performing when subjected to day-to-day scenarios.

PyTorch was used as the framework and the hyperparameters used for the model are as follows:

* An initial learning rate of 0.001 has been used.

* The batch size for the models is 64.

* The optimizer for the model is Adam.

* The loss function is Cross-Entropy Loss with SoftMax activation.

* The activation function for each layer is ReLU.

* The model has been trained for 5 epochs.

1. Experiment 1: Simple LeNET

This problem is an image classification problem and Convolutional Neural Networks are known to perform well in such Computer Vision tasks. For this, we have used a simple LeNET architecture. The hyperparameters are same as before and here we are not using any data augmentation or learning rate scheduler.

2. Experiment 2- Multi Layer Perceptron:

Our second experiment involves training a single hidden layer MLP with 128 neurons. The reason for using MLP is that our problem is a multi-class classification and MLP can be used as a multi logistic regression classification.

For this experiment we are using the initial learning rate with no data augmentation or learning rate scheduler to observe how the MLP performs on raw data.

3. Experiment 3: LeNET with Data Augmentation:

In real-life scenarios, it is not always the case that for sign language detection, the alignment of hand will be proper. Adding to that, people might use both their right and left hand for sign language.
To ensure that, the model performs well on such images as well, we have implemented data augmentation on our dataset.

Here we are using random horizontal flip and random rotation of ± 5°. However, the learning rate is fixed with initial learning rate.

4. Experiment 4: LeNET with Learning Rate Scheduler

After observing the results of the previous three experiments, we understood that the model learns a lot till the second epoch. To prevent overfitting and not to miss the optimum state, we have implemented a learning rate scheduler, such that after every 2 epochs, the learning rate gets reduced by 10%.

For this experiment, we haven't used data augmentation but just the learning rate scheduler.

After performing the four experiments, we used OpenCV to perform real-time detection of sign language.

V. Performance Analysis and Results

A. Comparison of different experiments

Table 1 documents the hyper-parameters used in each experiment, the training loss, the testing loss and the train time (in seconds).

Table 1- Comparison of different Experiments

Experiment	LeNET	MLP	Data Augmentation	Learning Rate Scheduler	Train Loss (NLL)	Test Loss (NLL)	Train Time (seconds)
1.	√				0.101	0.0162	1018
2.		√			0.143	0.0019	866
3.	√		√		0.256	0.131	1237
4.	√			√	0.100	0.0203	927

B. Results

1. Experiment 1:

The plot of learning curve and accuracy obtained for the training and testing set of experiment 1 is shown in Figure 3. Looking at the plots, we can see that the training loss reduces over 5 epochs. However, the testing loss shows a certain spike in the second epoch. The accuracy obtained in this experiment is around 94%.

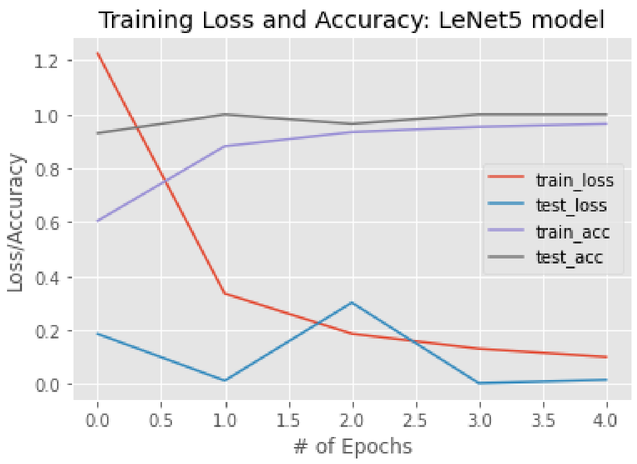


Figure 3: Learning Curve and Accuracy for Experiment 1

2. Experiment 2:

Figure 4 shows the plot of learning curve and accuracy of experiment 2. Observing the graph, we can see that the testing loss plateaus and then reduces in the later epochs.

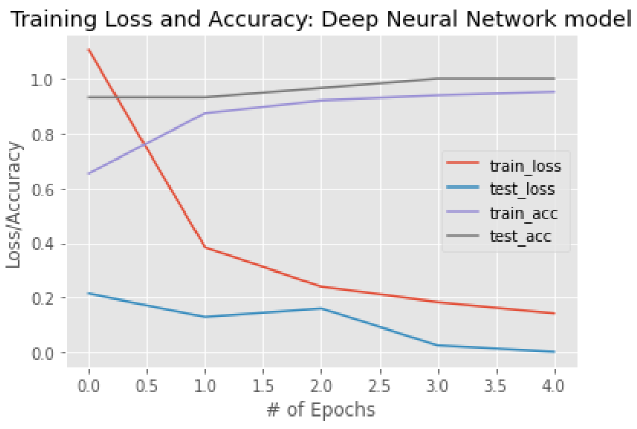


Figure 4: Learning and Accuracy Curve for Experiment 2

3. Experiment 3:

The plot of learning curve and accuracy for experiment 3 depicts that the training and testing loss is high, and the training accuracy is comparatively lower.

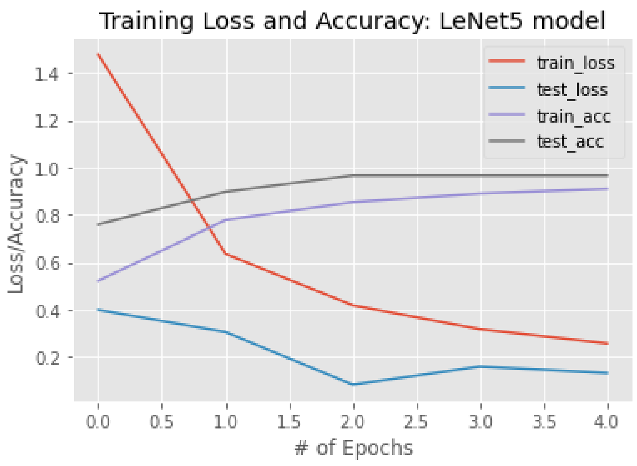


Figure 5: Learning Curve and Accuracy for Experiment 3

4. Experiment 4:

Experiment 4 performs the best compared to the rest of the experiments. The accuracy is about 96% for the training set. The training and testing loss is also less than other models. Adding to that, learning rate scheduler also ensures that we get the optimum value for convergence and the model doesn't overfit.

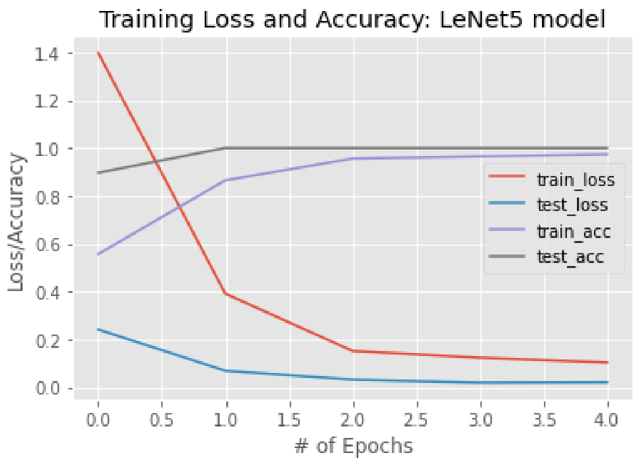
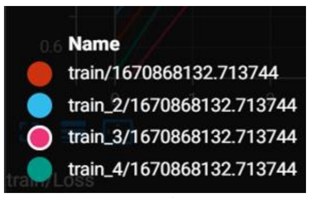


Figure 6: Learning Curve and Accuracy for Experiment 4

The TensorBoard Visualizations for all the four experiments for training and testing set is as follows:

1. TensorBoard Visualizations for Training



Name
● train/1670868132.713744
● train_2/1670868132.713744
● train_3/1670868132.713744
● train_4/1670868132.713744

Figure 7.a. Legend of each experiment

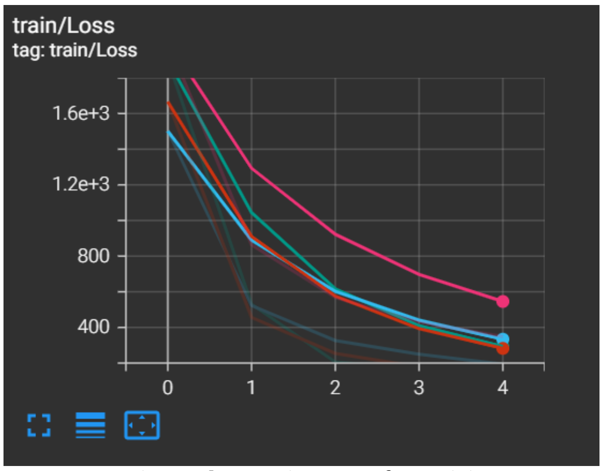


Figure 7.b. Learning Curve for Training Set

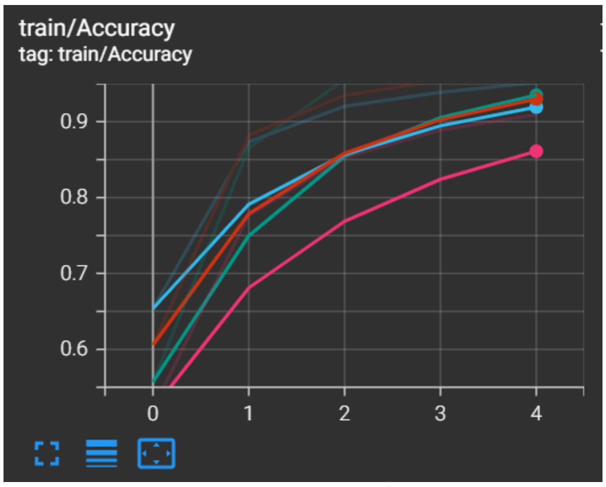
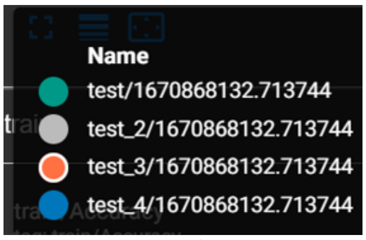


Figure 7.c. Accuracy Curve for Training Set

2. TensorBoard Visualizations for Testing



Name
● test/1670868132.713744
● test_2/1670868132.713744
● test_3/1670868132.713744
● test_4/1670868132.713744

Figure 8.a. Legend of each experiment

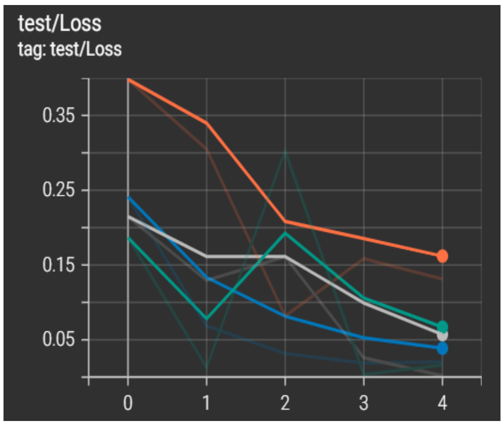


Figure 8.b. Learning Curve for Testing Set

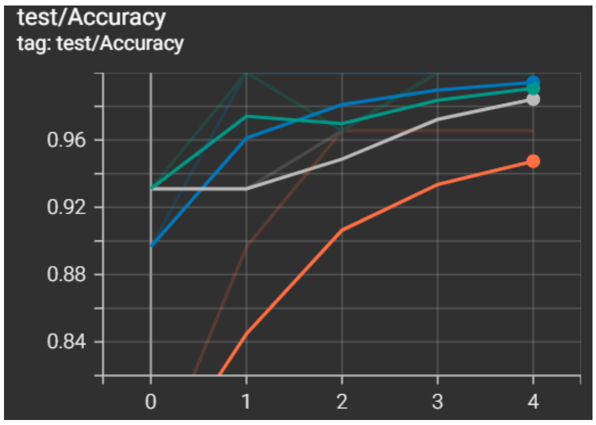
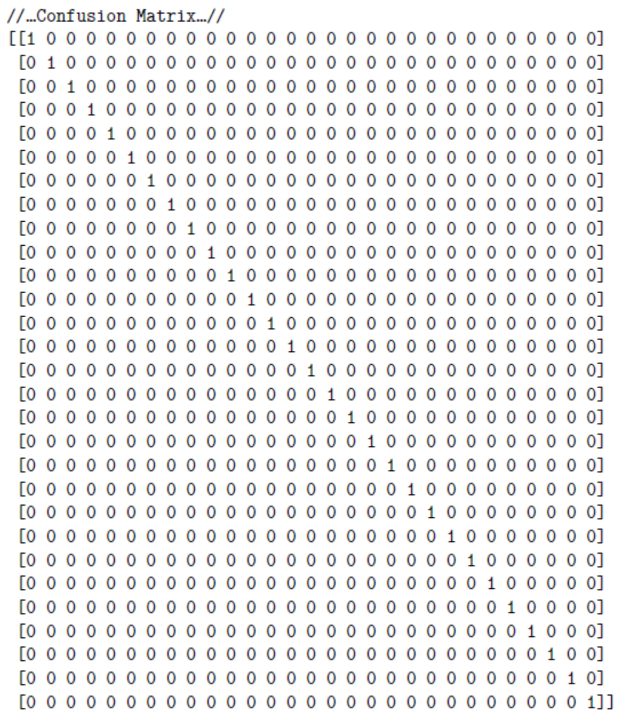


Figure 8.c. Accuracy for Testing Set

Experiment 4 provides the best result and the confusion matrix for the same has been shown below:



```
//…Confusion Matrix…//
[[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]]
```

Figure 9. Confusion matrix for Experiment 4

C. Challenges:

The biggest challenge that we faced was to implement the real-time video classification as we had to ensure that the input image captured by the web camera is same as the image format used by PyTorch.

VI. Conclusion and FutureScope

Classification of Sign Language is important as it helps us to break communication barrier between the hearing-impaired people and people with no disabilities. Using a LeNET architecture with learning rate decay is a great model for the simple classification of alphabets and certain special characters.

However, this is not enough. We need to implement a recommendation system which will add features like autocompletion of sentences. For example: if someone wants to say 'Goodbye', then as soon as the system detects the first two letters, it should prompt for the popularly used words starting with ‘G’ and followed by ‘O’. This will help a lot as it would reduce the effort required to spell out an entire word.

We code on CPU. So can this be transferred on embedded devices? The real-time inference ability of small sized Neural Networks with great accuracy can make it possible to be implemented on embedded devices which come with memory constraints. This looks quite feasible after implementing real-time ASL classification with saved model.

The authors have a curiosity that whether these trained model would work as good initialization on hand gesture recognition task. Also, will these models work for different skin toned hands? With no plain background?

REFERENCES

[1] R. Mitchell, T. Young, B. Bachleda, M. Karchmer. How Many People Use ASL in the United States?: Why Estimates Need Updating". Sign Language Studies (Gallaudet University Press.) 6 (3). ISSN 0302-1475. Retrieved November 27, 2012.

[2] T. Starner, J. Weaver and A. Pentland, "Real-time American sign language recognition using desk and wearable computer based video," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.20, no. 12, pp. 1371-1375, Dec. 1998, doi: 10.1109/34.735811.

[3] Izzah, A., & Suciati, N. (2014). Translation of sign language using generic fourier descriptor and nearest neighbour. International Journal on Cybernetics and Informatics, 3(1), 31–41. https://doi.org/10.5121/ijci.2014.3104.

[4] Y. Bajaj and P. Malhotra, "American sign language identification using hand trackpoint analysis", 2021.

[5] Architecture Construction: http://alexlenail.me/NN-SVG/index.html