# **MLP Model for ASL Sign Classification**

## **Project Description**

This project implements a Multi-Layer Perceptron (MLP) model to classify American Sign Language (ASL) hand signs into 29 categories. The model uses high-dimensional feature vectors as input and outputs the predicted class probabilities for each sample. The MLP serves as a baseline model for evaluating approaches to ASL sign classification.

## **Environment Requirements**

### **1. Programming Language**

* Python 3.8 or higher.

### **2. Required Libraries**

To run the code, you need to install the following dependencies:

pip install torch numpy matplotlib scikit-learn joblib

### **3. Hardware Requirements**

* **Processor**: Intel Core i5 or higher recommended.
* **RAM**: Minimum 8 GB (16 GB recommended for faster processing).
* **GPU**: Optional. If available, use an NVIDIA GPU with CUDA for accelerated training.

## **Data Preparation**

### **1. Input Files**

Place the following files in the working directory:

* feature\_data.npy: Contains the feature vectors of shape (94477,3258).
* feature\_labels.npy: Contains the corresponding labels (29 classes).

### **2. Preprocessing**

The following preprocessing steps are applied in the code:

1. **Standardization**:
   1. Features are normalized using StandardScaler to have zero mean and unit variance.
2. **Label Encoding**:
   1. Labels are converted to integer representations using LabelEncoder.

Preprocessed objects (scaler.pkl and label\_encoder.pkl) are saved for reuse.

## **Model Overview**

The MLP model consists of:

1. **Input Layer**: Accepts a feature vector of size *32583258*3258.
2. **Hidden Layers**:
   1. Two hidden layers, each with:
      1. 512 neurons.
      2. ReLU activation.
      3. Batch Normalization.
      4. Dropout (rate = 0.3).
3. **Output Layer**:
   1. 29 neurons with softmax activation to output probabilities for each class.

During training, learning rates from 0.1 to 0.0000001 were tested, with 0.0001 yielding the best results. Dropout rates of 0.1, 0.3, and 0.5 were evaluated, with 0.3 achieving the optimal balance between underfitting and overfitting.

## **How to Run**

### **1. Train the Model**

Run the following command to train the MLP model:

python train\_mlp.py

### **2. Output Files**

After training, the following files will be generated:

* **Trained Model**: best\_model.pth.
* **Scaler and Label Encoder**: scaler.pkl, label\_encoder.pkl.
* **Visualizations**: Loss, accuracy, and F1-score curves for training and validation.

### **3. View Results**

The final results, including training and validation accuracy, loss, and F1-score, are printed in the terminal and visualized using Matplotlib.

## **Model Performance**

* **Training Accuracy**: 69.31%.
* **Validation Accuracy**: 67.85%.
* **Validation F1-Score**: 0.6726.

The model achieves good convergence with minimal overfitting, as indicated by the steady loss and accuracy curves.

The validation F1-Score of 0.6726 suggests the model performs well across classes, though further improvement could address class imbalances.

## **Additional Notes**

* To modify hyperparameters such as learning rate, batch size, or the number of epochs, update the corresponding variables in train\_mlp.py.
* Ensure all dependencies are installed and input files are correctly placed in the working directory before running the code.