Overview

This competition introduces deep learning through a hands-on PyTorch challenge. You will work with hand landmark data representing 12 American Sign Language (ASL) gestures (alphabets). The landmarks are extracted using MediaPipe, which identifies key points on the hand from images or videos. To keep the focus on deep learning (since computer vision is not yet covered), the processed landmark data is provided directly. Each sample is a list of 21 hand landmarks, with each landmark described by three values: x (horizontal position), y (vertical position), and z (relative depth), together representing the 3D structure of the hand. This means there are 63 (21x3) features for each row.

The objective is to build a complete deep learning pipeline in PyTorch while exploring real-world challenges. Submissions are evaluated using the accuracy on a private test set, along with the number of model parameters. Simpler, more efficient models are preferred - adding complexity without meaningful performance improvements will be ranked lower. The focus is on learning how to design efficient deep learning models, not just maximizing scores.

Description

You are provided a dataset mapping hand coordinate locations to hand gesture (a letter from A-L). You need to build a deep learning model to make predictions given hand coordinate data.

In this competition, you are expected to build deep learning models using PyTorch. You are encouraged to try out various techniques of feature engineering.

My Solution:

After doing the usual data preprocessing and assigning my X and y, I used the label encoder to encode the output letters to numbers since neural networks work only with numbers.

I then converted my X and y arrays to tensors of the appropriate datatype.

I defined a class ASLDataset which answered how many samples there are and what the ith sample is.

I applied the ASLDataset class on the made tensors.

I then used DataLoader to load the data, and defined a batch size of 32.

I defined a neural network class consisting of pretty simple layers; Linear, ReLU, Linear, ReLU, and finally Linear.

For my loss function I used cross entropy loss, which penalizes wrong predictions heavily.

I used the adam optimizer with a learning rate of 0.001.

I trained the model on around 100 epochs (made multiple submissions seeing what gave me the best score, the more the epochs, the more the score would increase, but I also didn't want to overfit the model).

I defined an X_test, similarly converted that to an X_test_tensor, and applied the previously made classes on these.

Finally I made the predictions and converted the number outputs back to letters, and converted it to the submission format.