MAIS 202 - PROJECT DELIVERABLE 2

1. Problem Statement:

Our project is a computer vision application that leverages Convolutional Neural Networks (CNNs) to translate American Sign Language (ASL) into English in real-time. By capturing user input from a webcam, it interprets ASL gestures letter by letter

2. Data Preprocessing:

We have selected one of the two originally chosen datasets, containing 27,000 samples distributed across 27 classes representing the English alphabet, including a blank class. Each sample consists of a single feature, which is an image of the corresponding sign. We decided to replace the blank label with a space label due to the lack of the space label in the dataset. We also preprocessed the images by applying a filter that converts dark areas to black and brighter regions to white, a modification aimed at facilitating the training of convolutional neural networks on these images.

3. Machine Learning Model:

a. Framework and tools

We are using Pytorch to implement our model, and various other common libraries (numpy, matplotlib, opendatasets). We will likely follow a tutorial to implement our first model and as such will start with a LeNet architecture, with 2 layers of convolutions, ReLU, and maxpooling, then an FC layer, ReLU, FC and Softmax.

b. Justify decision-making

Since we haven't fully implemented out model and our dataset comes with a training/test set, we haven't come across many hyperparameter tuning or optimisation tasks yet, though we will be able to justify our hyperparameter settings using grid tests.

c. Validation

We will use the preexisting test split in our dataset, and continue with our confusion matrix plan, collecting precision, accuracy and F1 scores and evaluating our F1 score against a dummy classifier.

d. Challenges in implementing the model

One of the difficulties we faced when implementing the input portion of our program was the normalization of our data. In general, normalization is better for neural networks for the reasons seen in class, but when we were using tensorflow, we found that normalization must be an actual layer in the neural network, and it is relatively inconvenient to apply this later to the dataset directly and use that as our input. This is one of the reasons that motivated us to move from tensorflow/Keras to PyTorch, on top of the OOP and better fine-tuning capabilities the latter library offers.

4. Preliminary results:

For now, we have yet to have a substantial model to train on. However we have an idea of how to implement this model using PyTorch: using the Conv2D class from the torch.nn library, we can construct our own neural network model using the ideas seen in class. Furthermore, we know what hyperparameters we want to test on: the size of the kernels (i.e. the size of the filter we want to apply on each image after each layer) as well as their stride and padding, the number of convolution layers we want to apply, and the number of neurons for each layer. We will test all of these hyperparameters using grid tests.

5. Next steps:

We believe the next steps for this project is implementing the model class and testing it with different hyperparameters. We believe using Convolutional Neural Networks with PyTorch would be the best way to train an image recognition model at our level. One thing we may change in our model is modifying the input such that it is grayscale instead of RGB channels, because that might improve performance. In that case, the video input stream should also be modified.