Problem Statement: Implement and train a multi-layer perceptron (MLP) on the CIFAR-10 dataset with data augmentation, using a two hidden layer MLP model with 64 neurons each.

1. Download the CIFAR-10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html). Use the pickle library to load the dataset.

2. Implement the following image transformation methods:

(a) Image Enhancement:

Enhance each pixel i in the image x using the formula: ((i?min) / (max?min)) * 255.

Here, min and max are the minimum and maximum pixel values in the image x.

(b) Posterization of Image:

- Select a desired minimum and maximum pixel value in the range of [0-255].
- For each pixel i in image x:
  i. Calculate the range r by subtracting the selected minimum and maximum pixel values.
  ii. Get a divider for the colors using the formula: divider = 255 / r.
  iii. Get the level of colors by: i = i / divider.
  iv. Finally, apply the color palette on the pixel using: i = i + min.
- Ensure that the final image x has pixel values in the range of [0, 255].

(c) Random Rotate [-180°, 180°].

(d) Contrast & Horizontal Flipping:

- Change the contrast of the image with a factor of ?, randomly selected from the range (0.5, 2.0).
- Flip the image horizontally with a probability of 0.5.

To change the contrast of an image x with pixel values in the range [0, 255], use the following steps:

i. Adjust pixel values with the formula: x'(i, j, c) = ? * (x(i, j, c) ? 128) + 128.

ii. Clip pixel values in x' so that the final values are in the range [0, 255].

Note that ? < 1 will decrease the contrast of the image, while ? > 1 will increase the contrast.

For each transformation, define a Python function that takes an input image and returns the transformed image. Do not use any built-in functions (e.g., Pillow, sklearn, PIL, CV2). Fill any gaps created by transformation operations with zero values. Demonstrate each transformation by applying the functions to at least one example image.

3. Create the augmented training set using the transformation functions implemented in the previous part. Randomly select one of the four transformations for each image in the training set and apply it to that image. Combine the transformed images with the original training set to create the augmented training set. The number of examples in the augmented training set should be twice that of the unaugmented training set.

4. Use the provided feature extractor script (feature_extractor.py) on both the original (unaugmented) CIFAR-10 dataset and the augmented dataset to generate 1-dimensional input vectors. The feature extractor script should be used as-is, without modification.

Instructions for using feature_extractor.py:

- Refer to this page: https://pytorch.org/get-started/locally/ for installing the required dependencies of PyTorch. Note that PyTorch is already supported in environments like Google Colab or Kaggle.

- The feature extractor script accepts images of size (3×224×224) [Channel×Height×Width]. Resize the CIFAR images from (3×32×32) to (3 × 224 × 224) using image processing libraries like PIL or CV2.

- Pass the resized images to the feature extraction function of the BBResNet18 class to generate feature vectors.

- The feature extraction function expects each image to be a numpy.ndarray of dtype: numpy.float32 and shape: [None, 3, 224, 224], where None represents a variable size. It returns a numpy.ndarray of dtype: numpy.float32 and shape: [None, 512].

5. Implement a multi-layer perceptron (MLP) for classifying CIFAR-10 images. Use two hidden layers with 64 neurons each and the ReLU activation function. The input to this MLP will be the 1-dimensional vectors generated in the previous step.

6. Implement the back-propagation algorithm and use it to train the MLP model on:

(a) The original training set.

(b) The augmented training set.

Do not use built-in functions that perform back-propagation directly. Instead, write your own back-propagation algorithm.

7. Implement the following classification algorithms:

(a) SVM Classifier

(b) KNN Classifier

(c) Logistic Regression Classifier

(d) Decision Tree Classifier

You can use any built-in library functions for these ML classifiers.

8. Evaluate the performance of the above four classifiers and the trained MLP models on both the original (unaugmented) test set and the augmented test set for the top 5 classifications.