CSE343/ECE343: Machine Learning
Assignment-4 (CNN, K-means)
Max Marks: 25 (Programming: 15, Theory: 10) **Due Date**: 24/11/2024, 11:59 PM

Instructions

- Keep collaborations at high-level discussions. Copying/Plagiarism will be dealt with strictly.
- Late submission penalty: As per course policy.
- Your submission should be a single zip file **2020xxx_HW1.zip** (Where *2020xxx* is your roll number). Include **all the files (code and report with theory questions)** arranged with proper names. A single **.pdf report** explaining your codes with results, relevant graphs, visualization and solution to theory questions should be there. The structure of submission should follow:

  2020xxx_HW1
  |– code_rollno.py/.ipynb
  |– report_rollno.pdf
  |– (All other files for submission)

- Anything not in the report will **not** be graded.
- Remember to **turn in** after uploading on Google Classroom. No excuses or issues would be taken regarding this after the deadline.
- Start the assignment early. Resolve all your doubts from TAs in their office hours at least **two days before the deadline.**
- Your code should be neat and well-commented.
- **You have to do either Section B or C.**
- **Section A is mandatory.**

---

1. (10 points) **Section A (Theoretical)**

   (a) Consider the forward pass of a convolutional layer in a neural network architecture, where the input is an image of dimensions $M \times N$ (where $\min(M, N) \geq 1$) with P channels ($P \geq 1$) and a single kernel of size $K \times K$ ($1 \leq K \leq \min(M, N)$).

   (a) (1 point) Given a stride of 1 and no padding, determine the dimensions of the resulting feature map.

   (b) (1 point) Compute the number of elementary operations (multiplications and additions) required to compute a single output pixel in the resulting feature map.

(c) (3 points) Now, consider the scenario where there are Q kernels ($Q \geq 1$) of size $K \times K$. Derive the computational time complexity of the forward pass for the entire image in Big-O notation as a function of the relevant dimensions. Additionally, provide another Big-O notation assuming $\min(M, N) \gg K$.

(b) (5 points) Explain the Assignment Step and Update Step in the K-Means algorithm. Discuss any one method that helps in determining the optimal number of clusters. Can we randomly assign cluster centroids and arrive at global minima?

2. (15 points) **Section B (Scratch Implementation)**
You are tasked with implementing the KMeans clustering algorithm from scratch using Python. Use the Euclidean distance as the distance function where k is chosen as 2. The initial centroids for the 2 clusters are given as:

$$\mathbf{u}_1 = (3.0, 3.0)$$

$$\mathbf{u}_2 = (2.0, 2.0)$$

The matrix $\mathbf{X}$ consists of the following data points:

$$\mathbf{X} = \begin{pmatrix} 5.1 & 3.5 \\ 4.9 & 3.0 \\ 5.8 & 2.7 \\ 6.0 & 3.0 \\ 6.7 & 3.1 \\ 4.5 & 2.3 \\ 6.1 & 2.8 \\ 5.2 & 3.2 \\ 5.5 & 2.6 \\ 5.0 & 2.0 \\ 8.0 & 0.5 \\ 7.5 & 0.8 \\ 8.1 & -0.1 \\ 2.5 & 3.5 \\ 1.0 & 3.0 \\ 4.5 & -1.0 \\ 3.0 & -0.5 \\ 5.1 & -0.2 \\ 6.0 & -1.5 \\ 3.5 & -0.1 \\ 4.0 & 0.0 \\ 6.1 & 0.5 \\ 5.4 & -0.5 \\ 5.3 & 0.3 \\ 5.8 & 0.6 \end{pmatrix}$$

(a) Implement the k-means clustering algorithm from scratch. Ensure that your implementation includes:

   (a) (1 point) Initialization: Use the given centroids as starting points.

   (b) (2 points) Assignment: Assign each data point to the nearest centroid based on the Euclidean distance.

   (c) (2 points) Update: Recalculate the centroids after each assignment by computing the mean of all points assigned to each centroid.

   (d) (1 point) Convergence Check: Terminate the algorithm if centroids do not significantly change between iterations or after a maximum of 100 iterations. Use a convergence threshold of 1e-4.

(b) (2 points) Find the values of final centroids after the algorithm converges. Plot the two clusters at the start of the process and at the end.

(c) (2 points) Compare the results using the provided initial centroids versus using random initialization of centroids.

(d) (5 points) Determine the optimal number of clusters, M, using the Elbow method. Plot the Within-Cluster Sum of Squares (WCSS) against different values of k to find the elbow point. Randomly initialize M centroids, perform clustering and plot the resulting clusters

OR

3. (15 points) **Section C (Algorithm implementation using packages)** For this question, you are expected to work with the **CIFAR-10 dataset**. The CIFAR-10 dataset consists of 60,000 32x32 RGB images of 10 classes, with 6,000 images per class. You are expected to work with 3 classes from the available classes as per your choice. Hence, you should have roughly 18,000 images in your training curated dataset, with 15,000 images in the train and 3,000 in the test dataset, respectively. (Note: No additional marks will be provided for working with more classes than required.)

   1. (3 points) Data Preparation: Use PyTorch to load the CIFAR-10 dataset, perform a stratified random split in the ratio of 0.8:0.2 for the training and validation datasets. Here, the 15,000 images from the training dataset are split into train-val via 0.8:0.2 split, and 3,000 images (1,000 per class) are retained as the testing data from the original test dataset of CIFAR-10. Create a custom Dataset class for the data and create data loaders for all the dataset splits - train, val, and test.

   2. (0.5 points) Visualization: Load the dataset and visualize 5 images of each class from both the training and validation datasets.

   3. (2.5 points) CNN Implementation: Create a CNN architecture with 2 convolutional layers (using in-built PyTorch implementations) having a kernel size of 5 x 5, 16 channels, padding and stride of 1 for the first layer, and kernel size of 3 x 3, 32 channels, stride of 1, and padding of 0 for the second layer. Use max-pooling layers with a kernel size of 3 x 3 with a stride of 2 after the first convolutional layer and a kernel size of 3 x 3 with stride 3 after the second convolutional layer. After

the second max pooling layer, flatten out the output and add it to a multi-layer perceptron, with 16 neurons in the first layer and the classification head as the second layer. Use the ReLU activation function after each layer other than the last layer (the classification head layer).

4. (2.5 points) Training the model: Train the model using the cross-entropy loss function with Adam optimizer for 15 epochs. Log the training and validation loss and accuracy after each epoch. Save the trained models as .pth files, which are to be submitted along with the code for the assignment.

5. (1.5 points) Testing: Observe the training and validation plots for loss and accuracy and comment on your understanding of the results. Report the accuracy and F1-score on the test dataset. Plot the confusion matrix for the train, val and test dataset.

6. (3 points) Training an MLP: Create an MLP model with 2 fully connected layers, the first layer with 64 neurons and the second layer as the classification head. Flatten out the image before processing it into the MLP. Use a ReLU layer after the first fully connected layer, and use the cross-entropy loss function with adam optimizer to train the model for 15 epochs. Log the training and validation loss and accuracy after each epoch. Save the models as .pth files, which must be submitted along with the assignment.

7. (2 points) Infer and Compare: Compute the test accuracy and F1-score and plot the confusion matrix for the MLP model. Now, compare the results and plots obtained from both the models and comment on their performance and differences.

Note: During the demos, students will be expected to reproduce the evaluation results on the test dataset for both CNN and MLP models. Hence, it is critical to submit the .pth files of the trained models.