Machine Learning II

- Assignment #1 -

Course	Department	Student ID	Name
SOI1010	Automotive Engineering	2021048140	Dongmin Kim

September 30, 2025

1. Colab Notebook Link

https://colab.research.google.com/drive/1U7by7tjjO0DHKV_df6ZjV4NGnIxBwrmc?usp=sharing

2. Problems

(a) Implement k-NN (k = 5) using an iterative method

- In the iterative implementation, the k-NN algorithm directly computes the distance using an Python for loop. This is easy to understand and debug, because the logic follows directly from the mathematical definition of k-NN.

(b) Implement k-NN (k = 5) using broadcasting (vectorized)

- In the broadcasting implementation, Instead of computing the distance between the test example and each training example one by one, all elements differences can be computed simultaneously in a single tensor operation. The predicted label was identical to the iterative method in (a)

- The major advantage of this approach is efficiency. The actual runtime is significantly faster than before.

- Additionally, when using broadcasting in PyTorch, it is not strictly necessary to manually flatten the image tensors, since PyTorch can handle multi-dimensional arrays directly in distance computations.

- Broadcasting requires more memory than the iterative method because all elements are stored at once before summing. It could become problematic if both the number of dimension are large.

(c) Extend k-NN to multi-class classification over all digits (0–9)

- When extending (b) to classify all test images at once by broadcasting, we observed a near out-of-memory situation. The main cause was the implicit construction of a large intermediate tensor during distance computation *[num_test, num_train, D]* , which stresses RAM.

- Unlike the single example path, we must flatten input to align dimensions for broadcasting. In addition, *torch.bincount* becomes awkward because it only accepts 1D inputs, so we had to loop over test sample, which increased testing time.

(d) Issues found in (c) and improvements

Issues

- per-sample loop
- 3D broadcasting

Solution

- Vectorized voting: use *torch.scatter_add_* or *torch.nn.functional.one_hot(...).sum(dim=1)* to replace per-sample bincount.

- Avoid 3D broadcasting: compute pairwise distances with *torch.cdist* to directly obtain an [M, N] distance matrix without materializing [M, N, D].
  *Reference: https://docs.pytorch.org/docs/stable/generated/torch.cdist.html

- Since k-NN classification only depends on the relative order of distances, L2 and squared L2 yield identical nearest-neighbor results. Using the squared version can be slightly faster since it avoids the square root computation.

(e) Hyperparameters you can tune

- Number of Neighbors: k
- Distance metric: L1, L2, Cosine
- L1: $d_{L1}(x_i, x_j) = \sum_{k=1}^{D} |x_{i,k} - x_{j,k}|$
- L2: $d_{L2}(x_i, x_j) = \sqrt{\sum_{k=1}^{D} (x_{i,k} - x_{j,k})^2}$
- Cosine: $d_{\cos}(x_i, x_j) = 1 - \frac{x_i \cdot x_j}{\|x_i\| \|x_j\|}$

(f) Try alternative options and report observations

- We extended the baseline to compare three distance metrics (L1, L2, and Cosine) while sweeping k from 0 to 9
- All experiments used the same train/val split and identical preprocessing.

k	**L1 (%)**	**L2 (%)**	**Cos (%)**
0	10.08	10.08	10.08
1	96.77	97.47	97.97
2	95.78	96.78	97.55
3	**96.93**	**97.55**	97.97
4	96.48	97.27	98.00
5	96.47	97.27	**98.02**
6	96.43	97.27	97.85
7	96.60	97.23	97.85
8	96.42	97.20	97.73
9	96.37	97.12	97.67

- Best Metric: Cosine distance achieved the highest accuracy **(98.02%)**
- All three distance metrics performed similarly, with accuracy peaking around $k=5$
- Increasing k beyond 5 did not improve performance and sometimes led to minor accuracy drops
- L1 Distance showed slightly lower accuracy and took significantly longer computation time
- While evaluating all combinations of k=0~10 and three distance metrics (L1, L2, Cosine), the total TPU running time exceeded one hour, mainly due to the large intermediate tensors and slow L1 distance operations.

(g) Final test accuracy

- Based on the validation results in (f), we selected the best parameters that achieved the highest validation accuracy:
- k=5 for Cosine distance
- k=3 for L1 and L2 distances

Distance	**k**	**Validation Accuracy (%)**	**Test Accuracy (%)**
L1	3	96.93	96.09
L2	3	97.55	96.97
Cosine	5	98.02	**97.22**

- The best-performing configuration was Cosine distance with k=5, achieving a test accuracy of **97.22%.**