COMPUTER VISION AND PATTERN RECOGNITION [C]

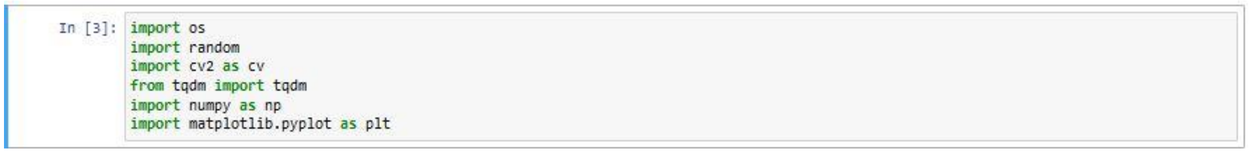
ASSIGNMENT-1[MID TERM]

Student Name: Shahriar Hossain Rafi

Student ID: 20-42528-1

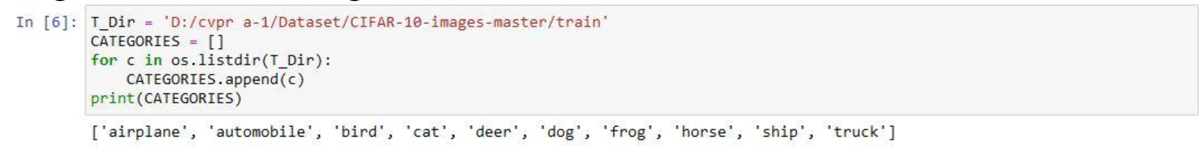
Submitted to: DR. DEBAJYOTI KARMAKER

1. **Importing the libraries that will be used to build our k-NN model.**



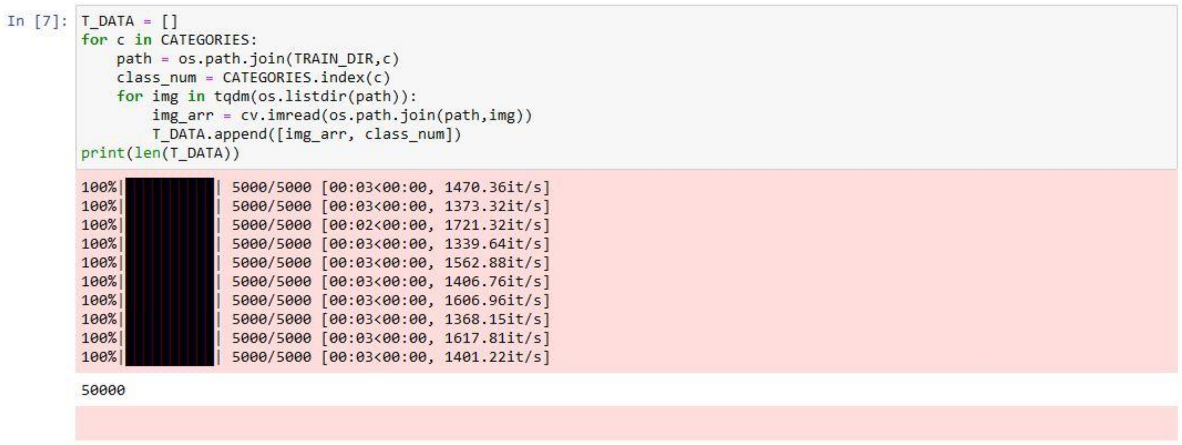
```
In [3]: import os
        import random
        import cv2 as cv
        from tqdm import tqdm
        import numpy as np
        import matplotlib.pyplot as plt
```

2. **Loading the CIFAR-10 training dataset.**



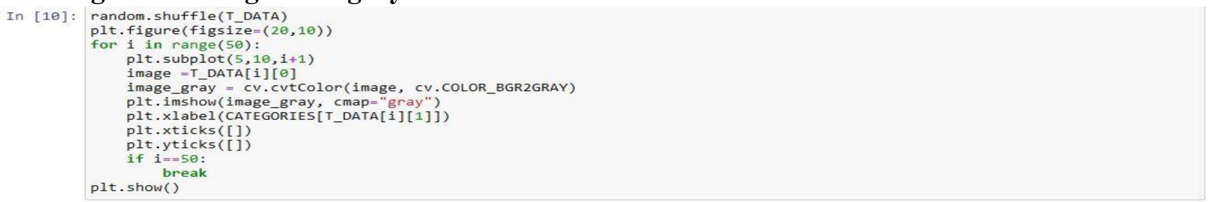
```
In [6]: T_Dir = 'D:/cvpr a-1/Dataset/CIFAR-10-images-master/train'
        CATEGORIES = []
        for c in os.listdir(T_Dir):
            CATEGORIES.append(c)
        print(CATEGORIES)

        ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

3. **Reading and storing every image from every class by recording them.**

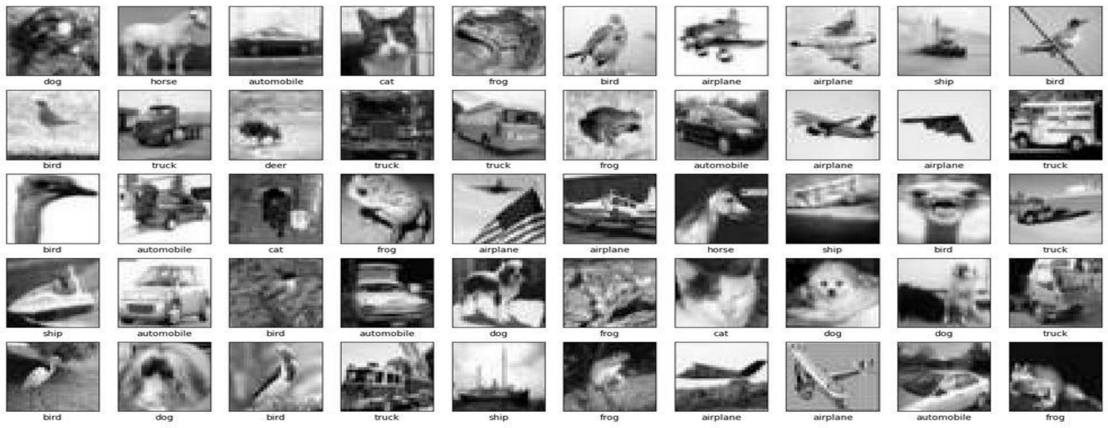


```
In [7]: T_DATA = []
        for c in CATEGORIES:
            path = os.path.join(TRAIN_DIR,c)
            class_num = CATEGORIES.index(c)
            for img in tqdm(os.listdir(path)):
                img_arr = cv.imread(os.path.join(path,img))
                T_DATA.append([img_arr, class_num])
        print(len(T_DATA))

        100%|██████████| 5000/5000 [00:03<00:00, 1470.36it/s]
        100%|██████████| 5000/5000 [00:03<00:00, 1373.32it/s]
        100%|██████████| 5000/5000 [00:02<00:00, 1721.32it/s]
        100%|██████████| 5000/5000 [00:03<00:00, 1339.64it/s]
        100%|██████████| 5000/5000 [00:03<00:00, 1562.88it/s]
        100%|██████████| 5000/5000 [00:03<00:00, 1406.76it/s]
        100%|██████████| 5000/5000 [00:03<00:00, 1606.96it/s]
        100%|██████████| 5000/5000 [00:03<00:00, 1368.15it/s]
        100%|██████████| 5000/5000 [00:03<00:00, 1617.81it/s]
        100%|██████████| 5000/5000 [00:03<00:00, 1401.22it/s]

        50000
```

4. **Converting all the images into grayscale.**



```
In [10]: random.shuffle(T_DATA)
         plt.figure(figsize=(20,10))
         for i in range(50):
             plt.subplot(5,10,i+1)
             image =T_DATA[i][0]
             image_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
             plt.imshow(image_gray, cmap="gray")
             plt.xlabel(CATEGORIES[T_DATA[i][1]])
             plt.xticks([])
             plt.yticks([])
             if i==50:
                 break
         plt.show()
```



5. Defining 5 folds of the data and split them as 1000 images per fold.

```
In [11]: f0 = T_DATA[0:1000]
         f1 = T_DATA[1000:2000]
         f2 = T_DATA[2000:3000]
         f3 = T_DATA[3000:4000]
         f4 = T_DATA[4000:5000]
```

6. Defining L1 and L2 distances

```
In [26]: def by_l1_dist(list):
             return list[2]["l1"]

         def by_l2_dist(list):
             return list[2]["l2"]
```

7. Training the model and calculating the accuracies for both L1 and L2 distance

```
In [27]: top_filter = 20
         def distance_calc(train_fold, valid_fold):
             l1_result = []
             l2_result = []
             for valid in tqdm(valid_fold):
                 temp_dist_list = []
                 for train in train_fold:
                     l1_dist = np.sum(np.abs(valid[0]-train[0]))
                     l2_dist = np.sqrt(np.sum(valid[0]-train[0]**2))
                     temp_dist_list.append([valid[1],  train[1], {"l1": l1_dist, "l2": l2_dist}])
                 temp_dist_list.sort(key=by_l1_dist)
                 l1_result.append(temp_dist_list[:top_filter])
                 temp_dist_list.sort(key=by_l2_dist)
                 l2_result.append(temp_dist_list[:top_filter])
             return [l1_result, l2_result]
```

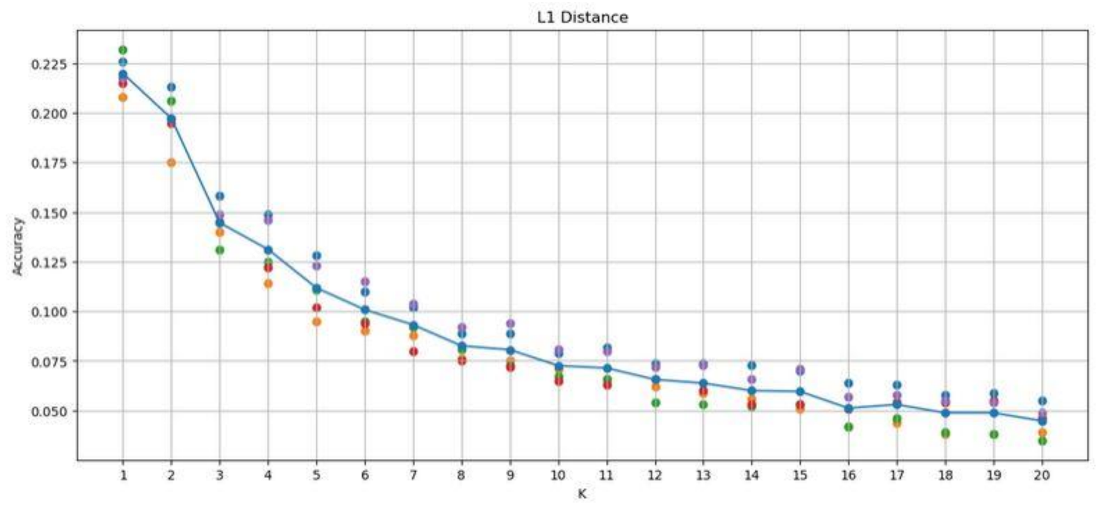
```
In [28]: k_range = 20
         def cal_accuracy(dist_result, dist_term):
             k_accuracies = []
             for k in range(1, k_range+1):
                 img_accuracy = 0
                 for valid_img in dist_result:
                     nn = valid_img[:k]
                     same_class = [n for n in nn if n[0] == n[1]]
                     same_class_len = len(same_class)
                     if k % 2 != 0:
                         if ((k-1) / 2) < same_class_len:
                             img_accuracy += 1
                     else:
                         diff_class = [n for n in nn if n[0] != n[1]]
                         if same_class_len > len(diff_class):
                             img_accuracy += 1
                         elif same_class_len == len(diff_class): # tie
                             same_class_dist = sum([n[2][dist_term] for n in same_class])
                             diff_class_dist = sum([n[2][dist_term] for n in diff_class])
                             if same_class_dist > diff_class_dist:
                                 img_accuracy += 1
                 k_accuracies.append(img_accuracy/len(dist_result))
             return k_accuracies
```

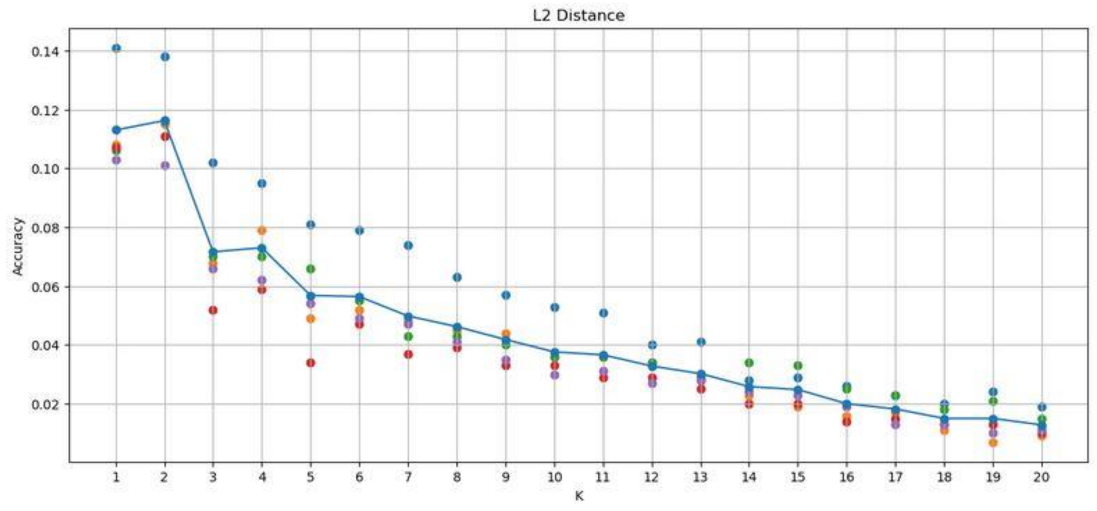
```
In [29]: dist_by_fold = []
         import math
         for i in range(5):
             if i==0:
                 train = f1+f2+f3+f4
                 validation = f0
             elif i==1:
                 train = f0+f2+f3+f4
                 validation = f1
             elif i==2:
                 train = f1+f0+f3+f4
                 validation = f2
             elif i==3:
                 train = f1+f2+f0+f4
                 validation = f3
             elif i==4:
                 train = f1+f2+f3+f0
                 validation = f4

             dist_by_fold.append(distance_calc(train, validation))

100%|██████████| 1000/1000 [01:51<00:00,  8.97it/s]
100%|██████████| 1000/1000 [01:54<00:00,  8.73it/s]
100%|██████████| 1000/1000 [01:54<00:00,  8.71it/s]
100%|██████████| 1000/1000 [01:54<00:00,  8.73it/s]
100%|██████████| 1000/1000 [01:53<00:00,  8.82it/s]
```

```
In [30]: accuracies = []

         for result in dist_by_fold:
             l1_accuracy = cal_accuracy(result[0], "l1")
             l2_accuracy = cal_accuracy(result[1], "l2")
             accuracies.append([l1_accuracy, l2_accuracy])
```

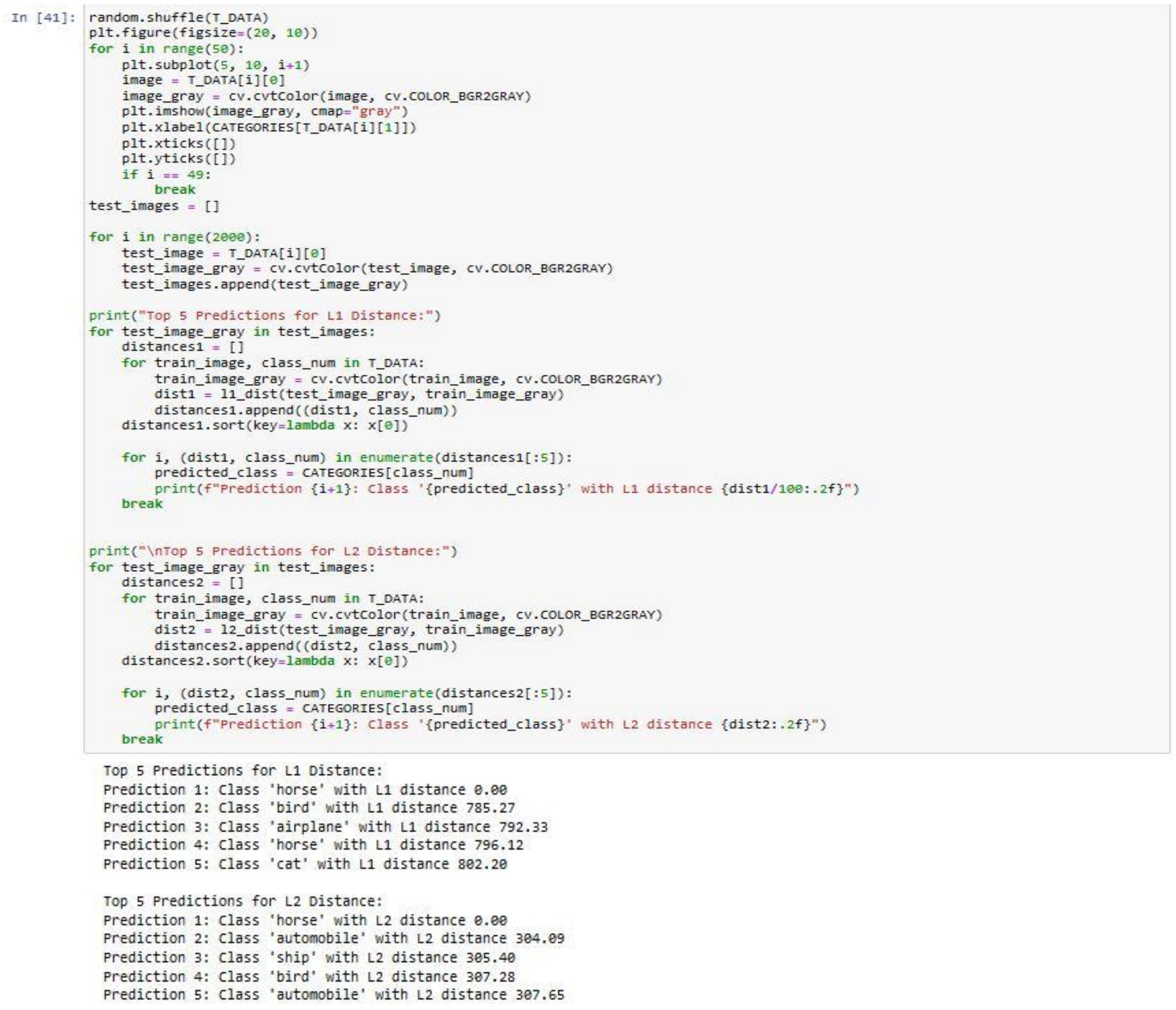
8. Plotting the results of L1 and L2 distance accuracies.

```
In [31]: plt.figure(figsize=(14, 6))

         for fold in accuracies:
             plt.scatter(list(range(1, k_range+1)), fold[0])
         arr = []
         for i in range(k_range):
           arr.append([fold[0][i] for fold in accuracies])
         trend = [np.mean(a) for a in arr]
         plt.errorbar(list(range(1, k_range+1)), trend,fmt='-o')
         plt.title('L1 Distance')
         plt.xticks(list(range(1, k_range+1)))
         plt.grid(True)
         plt.xlabel('K')
         plt.ylabel('Accuracy')
         plt.show()

         plt.figure(figsize=(14, 6))

         for fold in accuracies:

             plt.scatter(list(range(1, k_range+1)), fold[1])
         arr = []
         for i in range(k_range):
           arr.append([fold[1][i] for fold in accuracies])
         trend = [np.mean(a) for a in arr]
         plt.errorbar(list(range(1, k_range+1)), trend,fmt='-o')
         plt.title('L2 Distance')
         plt.xticks(list(range(1, k_range+1)))
         plt.grid(True)
         plt.xlabel('K')
         plt.ylabel('Accuracy')
         plt.show()
```





9. **Predicting the top 5 images by L1 and L2 distance. Each Prediction will have the class name that the KNN model predicted and the distance it calculated for that value**



```
In [41]: random.shuffle(T_DATA)
         plt.figure(figsize=(20, 10))
         for i in range(50):
             plt.subplot(5, 10, i+1)
             image = T_DATA[i][0]
             image_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
             plt.imshow(image_gray, cmap="gray")
             plt.xlabel(CATEGORIES[T_DATA[i][1]])
             plt.xticks([])
             plt.yticks([])
             if i == 49:
                 break
         test_images = []

         for i in range(2000):
             test_image = T_DATA[i][0]
             test_image_gray = cv.cvtColor(test_image, cv.COLOR_BGR2GRAY)
             test_images.append(test_image_gray)

         print("Top 5 Predictions for L1 Distance:")
         for test_image_gray in test_images:
             distances1 = []
             for train_image, class_num in T_DATA:
                 train_image_gray = cv.cvtColor(train_image, cv.COLOR_BGR2GRAY)
                 dist1 = l1_dist(test_image_gray, train_image_gray)
                 distances1.append((dist1, class_num))
             distances1.sort(key=lambda x: x[0])

             for i, (dist1, class_num) in enumerate(distances1[:5]):
                 predicted_class = CATEGORIES[class_num]
                 print(f"Prediction {i+1}: Class '{predicted_class}' with L1 distance {dist1/100:.2f}")
             break


         print("\nTop 5 Predictions for L2 Distance:")
         for test_image_gray in test_images:
             distances2 = []
             for train_image, class_num in T_DATA:
                 train_image_gray = cv.cvtColor(train_image, cv.COLOR_BGR2GRAY)
                 dist2 = l2_dist(test_image_gray, train_image_gray)
                 distances2.append((dist2, class_num))
             distances2.sort(key=lambda x: x[0])

             for i, (dist2, class_num) in enumerate(distances2[:5]):
                 predicted_class = CATEGORIES[class_num]
                 print(f"Prediction {i+1}: Class '{predicted_class}' with L2 distance {dist2:.2f}")
             break
```

```
Top 5 Predictions for L1 Distance:
Prediction 1: Class 'horse' with L1 distance 0.00
Prediction 2: Class 'bird' with L1 distance 785.27
Prediction 3: Class 'airplane' with L1 distance 792.33
Prediction 4: Class 'horse' with L1 distance 796.12
Prediction 5: Class 'cat' with L1 distance 802.20

Top 5 Predictions for L2 Distance:
Prediction 1: Class 'horse' with L2 distance 0.00
Prediction 2: Class 'automobile' with L2 distance 304.09
Prediction 3: Class 'ship' with L2 distance 305.40
Prediction 4: Class 'bird' with L2 distance 307.28
Prediction 5: Class 'automobile' with L2 distance 307.65
```

Discussion:

The performance of Manhattan (L1) and Euclidean (L2) distances was compared based on the average accuracy values obtained from the 5-fold cross-validation using the CIFAR-10 dataset, which consists of 60,000 32x32x3 color images in 10 different classes. Aim was to determine which distance calculation technique is most appropriate for the given gray-scale dataset. According to the analysis, the L1 distance exceeded the L2 distance in terms of accuracy. The Manhattan (L1) and Euclidean (L2) distance comparisons illustrated the significance of taking dataset characteristics and feature nature into account when choosing a distance calculations approach.