Homework laboratory 1

Exercise 1

The size of the four vectors generated after loading CIFAR-10 dataset:

```
    print(x_train.shape)
    print(y_train.shape)
    print(x_test.shape)
    print(y_test.shape)
```

x_train: (50000, 32, 32, 3)

y_train: (50000, 1)

x_test: (10000, 32, 32, 3)

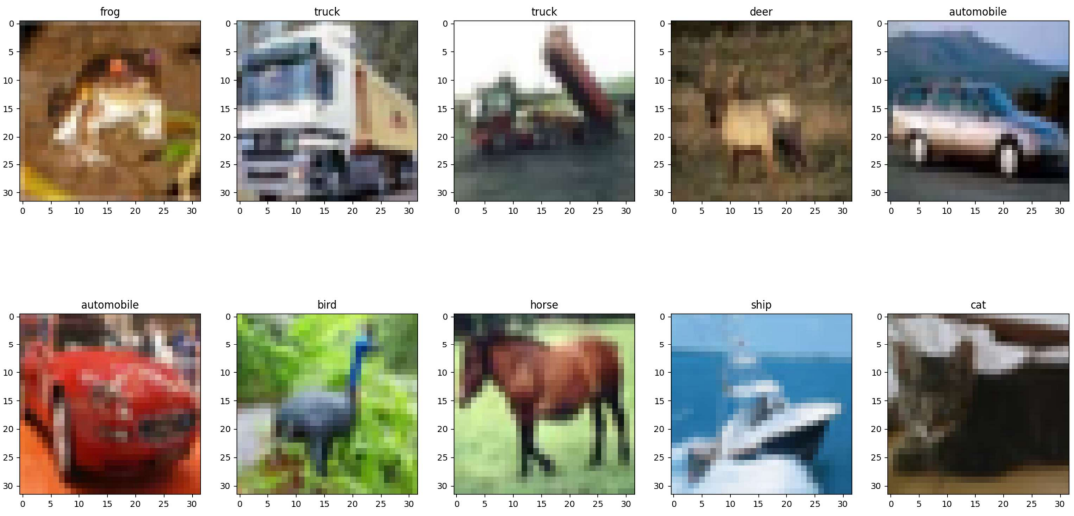
y_test: (10000, 1)

Exercise 2

Visualize the first 10 images from the testing dataset with their associated labels

```
    for i in range(10):
        # define subplot
        ax = plt.subplot(2, 5, 1 + i)
        # plot raw pixel data
        plt.imshow(x_train[i], cmap=plt.get_cmap('gray'))
        ax.set_title(dict_classes[y_train[i][0]])

    plt.show()
```



Exercise 3

accuracy: 29.0%

```
def predictLabelNN(x_train_flatten, y_train, img):

    predicted_label_index = -1
    scoreMin = 100000000

    for idx, imgT in enumerate(x_train_flatten):

        difference = abs(np.subtract(img, imgT))
        difference_L2 = difference ** 2

        score = np.sqrt(np.sum(difference_L2))

        if score < scoreMin:
            predicted_label_index = idx
            scoreMin = score
```

We will use this form of predict function. It uses a L2 difference for training.

Exercise 4 and 5

```
def most_frequent(l):
    count_labels_dict = {}

    for item in l:
        if item[1] not in count_labels_dict:
            count_labels_dict[item[1]] = 1
        else:
            count_labels_dict[item[1]] += 1

    return max(count_labels_dict, key=lambda k: count_labels_dict[k])

def predictLabelKNN(x_train_flatten, y_train, img, k=3, diff='L2'):

    predictedLabel = -1
    predictions = []  # list to save the scores and associated labels as pairs  (score, label)
    score = 0

    for idx, imgT in enumerate(x_train_flatten):

        difference = abs(np.subtract(img, imgT))

        if diff == 'L2':
            difference = difference ** 2

        score = np.sum(difference)

        predictions.append([score, y_train[idx][0]])

    predictions = sorted(predictions, key=lambda x: x[0])
```

```
    top_k_predictions = predictions[:k]
    print(top_k_predictions)


    predictedLabel = most_frequent(top_k_predictions)

    return predictedLabel


def do_every_combination(N, x_train_flatten, y_train, x_test_flatten, y_test, diff_type='L1'):
    info = {}

    for i in [1, 3, 5, 10, 20, 50]:
        info[i] = {}

        info[i]['correct_predicted'] = 0
        info[i]['last_predicted'] = 0
        info[i]['accuracy'] = 0

    for idx, img in enumerate(x_test_flatten[0:N]):

        print(f"Make a prediction for image {idx}")


        # TODO - Application 1 - Step 3 - Call the predictLabelNN function
        for key in info:
            info[key]['last_predicted'] = predictLabelKNN(x_train_flatten, y_train, img, k=key
, diff=diff_type)



        # TODO - Application 1 - Step 4 - Compare the predicted label with the groundtruth (th
e label from y_test).
        #  If there is a match then increment the contor numberOfCorrectPredictedImages

        for key in info:
            if info[key]['last_predicted'] == y_test[idx]:
                info[key]['correct_predicted'] += 1


    with open(f"./results_{diff_type}.txt", "w") as f:

        for key in info:
            info[key]['accuracy'] = N * info[key]['correct_predicted'] / 100
            f.write(f"{diff_type} - k={key} ========== {info[key]['accuracy']}\n")

    return info

def exercises_4_5(diff_type='L1'):
    (x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

    print(x_train.shape)
```

```
    print(y_train.shape)
    print(x_test.shape)
    print(y_test.shape)

    x_train_flatten = x_train.reshape(x_train.shape[0], 32 * 32 * 3)
    x_test_flatten = x_test.reshape(x_test.shape[0], 32 * 32 * 3)

    N = 100

    do_every_combination(N, x_train_flatten, y_train, x_test_flatten, y_test, diff_type=diff_t
ype)

if __name__ == '__main__':
    # main()
    exercises_4_5('L1')
```

We just have to change the argument of the last function call from 'L1' to 'L2' for exercise 5.

This script takes every photo, and it passes it through each KNN classifier (1, 3, 5, 10, 20, 50), then writes the output to files.

Output from files:

L1 - k=1 ========= 23.0

L1 - k=3 ========= 23.0

L1 - k=5 ========= 26.0

L1 - k=10 ========= 33.0

L1 - k=20 ========= 33.0

L1 - k=50 ========= 35.0

L2 - k=1 ========= 29.0

L2 - k=3 ========= 27.0

L2 - k=5 ========= 24.0

L2 - k=10 ========= 24.0

L2 - k=20 ========= 24.0

L2 - k=50 ========= 25.0

We can see that when using L2, the accuracy actually stayed the same or even decreased when we increased the number of neighbors.

I've changed the `most_frequency` function a bit since I understand it better this way, I hope this is not a problem. Basically, we create a key for each new discovered label and initiate the value with 1. Each time we encounter this label again, we just increment the value of its key.

This process could’ve been sped up with the multiprocessing library.