3. **[16pts] kNN vs. Logistic Regression.** In this problem, you will compare the performance and characteristics of two classifiers: k-Nearest Neighbors and Logistic Regression. You will complete the provided code in `q2/` and experiment with the completed code. You should understand the code instead of using it as a black box.

   3.1. **k-Nearest Neighbors.** Use the supplied kNN implementation to predict labels for `mnist_valid`, using the training set `mnist_train`.

   (a) **[3pts]** Implement the function `run_knn` in `run_knn.py` that runs kNN for different values of $k \in \{1, 3, 5, 7, 9\}$ and plots the classification accuracy on the validation set as a function of k. The classification accuracy is the number of correctly predicted data points divided by the total number of data points. Include the plot in the write-up.

   (b) **[2pts]** Choose a value of k (strictly positive integer) and justify your choice. Let's denote the chosen value by k^*. Report the validation and test accuracies of KNN for k^*. Also, report the validation and test accuracies of KNN for $k^* + 2$ and $k^* - 2$ (report the latter if your value of k^* is greater than two).
   How do the test and validation accuracies change as we change the value of k?
   In general, you shouldn't peek at the test set multiple times, but we do this for this question as an illustrative exercise.

   3.2. **Logistic Regression.** Read the provided code in `run_logistic_regression.py` and `logistic.py`. You need to implement the logistic regression model, where the cost is defined as:

$$\mathcal{J} = \sum_{i=1}^{N} \mathcal{L}_{\text{CE}}(y^{(i)}, t^{(i)}) = \sum_{i=1}^{N} \Big(-t^{(i)} \log y^{(i)} - (1 - t^{(i)}) \log(1 - y^{(i)}) \Big),$$

   where N is the total number of data. Note that the cost function is the total loss instead of the average loss.

   (a) **[4pts]** Implement functions `logistic_predict`, `evaluate`, and `logistic` in the file `logistic.py`.

   (b) **[5pts]** Complete the missing parts in the function `run_logistic_regression`. Run the code on both `mnist_train` and `mnist_train_small`. Check whether the value returned by `run_check_grad` is small to make sure your implementation in part (a) is correct.

   Experiment with the hyper-parameters for the learning rate, the number of iterations, and the initial values of the weights. If you have a smaller learning rate, your model will take longer to converge.

   If you get `NaN/Inf` errors, you may try to reduce your learning rate or initialize with smaller weights. Report the best hyper-parameter settings you found and the final cross entropy and classification accuracy on the training, validation, and test sets. You should only compute the test error once you have selected your best hyper-parameter settings using the validation set.

(c) **[2pts]** Examine how the cross entropy changes as training progresses. Generate and report 2 plots, one for each of `mnist_train` and `mnist_train_small`. In each plot, you need show two curves: one for the training set and one for the validation set. Run your code several times and observe if the results change. If they do, how would you choose the best parameter settings?