```python
import numpy as np
import glob
from PIL import Image
from sklearn.model_selection import KFold
from scipy.stats import mode
import matplotlib.pyplot as plt


def load_images(path: str = "./tiny/*.png") -> np.array:
    """Load images.
    Args:
        path (str, optional): path ot find all images. Defaults to "./tiny/*.png".
    Returns:
        np.array: array of (nums_img * 32 * 32 * 3).
    """
    img_paths = glob.glob(path)
    imgs = np.array([np.array(Image.open(p)) for p in img_paths])
    return imgs


def load_labels(path: str = "./tiny/labels.txt") -> np.array:
    """Load labels and expand it.
    Args:
        path (str, optional): path to the labels file. Defaults to "./tiny/labels.txt".
    Returns:
        np.array: array of labels (0 - 5)
    """
    with open(path) as file:
        lines = file.readlines()
        ranges = len(lines)
        lines = [line.rstrip().split("-") for line in lines]
        lines = [int(item) for sublist in lines for item in sublist]
    labels = np.zeros((max(lines)))
    for i in range(ranges):
        labels[lines[i * 2] - 1 : lines[i * 2 + 1]] = i
    return labels


def knn(n_neighbors:int, images:np.array, labels:np.array, train_ids:np.array, test_ids:np.array)
->int:
    """Perform knn and return the accuracy.
    Args:
        n_neighbors (int): the number of neighbors.
        images (np.array): images.
        labels (np.array): labels.
        train_ids (np.array): ids of all training data.
        test_ids (np.array): ids of all testing data.
    Returns:
        int: accuracy.
    """

    nums_train = len(train_ids)
    nums_test = len(test_ids)

    accuracy = 0

    for test_id in test_ids:
        test_img = images[test_id]
        distance = abs(images[train_ids, ...] - np.tile(test_img, (nums_train, 1, 1, 1)))
        distance = distance.reshape(nums_train, -1).sum(1)
```

```
        #Find the index of lowest values if the array is sorted.
        min_neighbors_ids = np.argsort(distance)[0:n_neighbors]

        min_neighbors_labels = labels[train_ids][min_neighbors_ids]

        vote = mode(min_neighbors_labels)[0]
        accuracy += int(vote == labels[test_id])

    return accuracy / nums_test


def main():
    #Load images and labels
    images = load_images()
    labels = load_labels()

    #Split data into 10 folds.
    n_splits = 10
    partition = list(KFold(n_splits=n_splits, shuffle=True).split(images))

    #Set n_neighbors
    n_neighbors = 15
    neighbors_accuracy = []

    #Iteration through 1 to 15 neighbors.
    for neighbors in range(1, n_neighbors + 1):
        split_accuracy = 0

        #Per each neighbors, iterate through the first 9 folds.
        for split in range(0, n_splits-1):
            train_ids = partition[split][0]
            test_ids = partition[split][1]
            split_accuracy += knn(neighbors, images, labels, train_ids, test_ids)

        split_accuracy /= n_splits - 1
        neighbors_accuracy += [split_accuracy]


    #Plot the average accuracy for the first 9 folds.
    fig, ax = plt.subplots(figsize=(12.80, 7.20))
    ax.bar(
        range(1, n_neighbors + 1), [float("%.5f" % acc) for acc in neighbors_accuracy]
    )
    ax.bar_label(ax.containers[0])
    ax.set_xlabel("Number of Neighbors")
    ax.set_ylabel("Average Accuracy")
    ax.set_title("Average Accuracy of All Possible Neighbors")
    fig.savefig("average_accuracy.png", dpi=300)

    #Pick the best neighbor
    best_n_neighbors = np.argmax(neighbors_accuracy) + 1

    #Find the average accuracy of the best neigbor with the last fold.
    train_ids = partition[n_splits - 1][0]
    test_ids = partition[n_splits - 1][1]

    test_accuracy = knn(best_n_neighbors, images, labels, train_ids, test_ids)

    print(f"Best n_neighbors: {best_n_neighbors}")
    print(f"Accuracy: {test_accuracy:.4f}")
```

```
if __name__ == "__main__":
    main()
```

4. Leave one-fold aside for testing and the remaining 9 folds for training and validation. Explain how you did that.
Ans: Use KFold function from sklearn to split the data into 10 folds. KFold returns 10 tuples of training_ids and testing_ids. testing_ids is around 10% of the entire data. Use the first 9 folds to find the best n_neighbors. Then, use the best n_neighbors and the last fold to calculate the accuracy.