ECE194N HW 1

KNN Report

February 8, 2019

Student: Erik Rosten
Perm Number: 7143571
Email: erosten@ucsb.edu

Department of Electrical and Computer Engineering, UCSB

Code

My code for this programming problem is below

```
import numpy as np
import cifar_loader
import matplotlib.pyplot as plt
import os



def split_sets(all_images, all_classes, labels_to_keep):
   images = []
   classes = []
   for i in range(labels_to_keep.shape[0]):
      indices = np.where(all_classes == labels_to_keep[i])[0]
      images.append(all_images[indices])
      classes.append(all_classes[indices])

   images = np.array(images)
   images = images.reshape([-1,32,32,3])
   classes = np.array(classes)
   classes = classes.reshape([-1,])
   return images, classes

def split_class_names(all_class_names, class_numbers):
   class_names = []
   for i in range(class_numbers.shape[0]):
      class_names.append(all_class_names[class_numbers[i]])
   return np.array(class_names)

def view_examples(train_imgs, train_cls, labels_to_keep, class_names):
   print(class_names)
   plt.figure(1)
   for i in range(labels_to_keep.shape[0]):
      label_class_name = class_names[i]
      rand_integer = np.random.randint(0,4999)
      # grab a random example index
      index = np.where(train_cls == labels_to_keep[i])[0][rand_integer]
      img_example = train_imgs[index]

      ax = plt.subplot(1,labels_to_keep.shape[0],i + 1)
      ax.axis('off')
      imgplot = plt.imshow(img_example)
      ax.set_title(label_class_name)
   plt.show()


def rgb2gray_average(rgb_imgs):
   gray_imgs = np.zeros((rgb_imgs.shape[0],32,32))
   for i in range(rgb_imgs.shape[0]):
      r, g, b = rgb_imgs[i,:,:,0], rgb_imgs[i,:,:,1], rgb_imgs[i,:,:,2]
      gray_imgs[i,:,:] = (r + g + b) / 3
   return gray_imgs
```

```
def rgb2gray_luminosity(rgb_imgs):
   gray_imgs = np.zeros((rgb_imgs.shape[0],32,32))
   for i in range(rgb_imgs.shape[0]):
      r, g, b = rgb_imgs[i,:,:,0], rgb_imgs[i,:,:,1], rgb_imgs[i,:,:,2]
      gray_imgs[i,:,:] = 0.21 * r + 0.72 * g + 0.07 * b
   return gray_imgs

def euclidean_dist(img1, img2):
   return np.linalg.norm(img1-img2)

def ssd(img1, img2):
   return np.sum((img1-img2)**2)

def compute_dists(train_imgs, test_imgs):
   num_test = test_imgs.shape[0]
   num_train = train_imgs.shape[0]
   dists = np.zeros((num_test, num_train))
   for i in range(num_test):
      print(i)
      for j in range(num_train):
         dists[i,j] = euclidean_dist(train_imgs[j,:], test_imgs[i,:])
   return dists

def find_nearest_neighbors(k, dists, train_cls):
   nn_distances = np.zeros((dists.shape[0],k))
   nn_classes = np.zeros((dists.shape[0],k))
   nn_classes_indices = np.zeros((dists.shape[0],k))
   for i in range(dists.shape[0]):
      idx = np.argpartition(dists[i,:], k, axis = 0)[:k]
      nn_distances[i,:] = dists[i,idx]
      nn_classes[i,:] = train_cls[idx]
      nn_classes_indices[i,:] = idx
   return nn_distances, nn_classes.astype(int), nn_classes_indices.astype(int)

def compute_error_rate(nn, test_cls , nn_cls):
   acc_count = 0
   total_test_imgs = nn.shape[0]
   k = nn.shape[1]

   for i in range(total_test_imgs):
      unique_labels = np.unique(nn_cls[i])
      label_count = 0
      label = np.max(unique_labels) + 1
      for j in range(unique_labels.shape[0]):
         num_occurences = np.count_nonzero(nn_cls[i] == unique_labels[j])
         if (num_occurences > label_count):
            label_count = num_occurences
            label = unique_labels[j]
         elif(num_occurences == label_count):
            new_indices = np.where(nn_cls[i] == unique_labels[j])[0]
            old_indices = np.where(nn_cls[i] == label)[0]
            new_avg = np.average(nn[i,new_indices])
            old_avg = np.average(nn[i,old_indices])
```

```
                if (new_avg < old_avg):
                    label = unique_labels[j]

        if (test_cls[i] != label):
            acc_count = acc_count + 1

    return acc_count / total_test_imgs

def plot_nearest_neighbors(labels_to_keep, dists, train_cls, test_cls, test_imgs, train_imgs, class_names):

    for i in range(labels_to_keep.shape[0]):
        label = labels_to_keep[i]
        nn_dists, nn_cls, nn_cls_indices = find_nearest_neighbors(5, dists, train_cls)
        label_indices = np.where(test_cls == label)[0]
        ind = np.random.randint(0, label_indices.shape[0] - 1)
        label_index = label_indices[ind]
        nn_img_indices = nn_cls_indices[label_index,:]
        nn_cls = nn_cls[label_index]
        nn_dists = nn_dists[label_index]

        plt.figure(1)
        ax1 = plt.subplot(2,3,1)
        ax1.axis('off')
        imgplt1 = plt.imshow(test_imgs[label_index])
        ax1.set_title('Test Image ({})'.format(class_names[i]), size=10)

        for j in range(5):
            ax2 = plt.subplot(2,3,(j+2))
            ax2.axis('off')
            nn_class = class_names[np.where(labels_to_keep == nn_cls[j])[0][0]]
            imgplt2 = plt.imshow(train_imgs[nn_img_indices[j]])
            ax2.set_title('NN: {} , dist: {:0.2f}, class: {}'.format(j+1, nn_dists[j],nn_class), size=7)

        plt.show()

def plot_k_error_rates(k_array, dists, train_cls, test_cls):
    e_rates = np.zeros(k_array.shape)
    for k in range(k_array.shape[0]):
        nn, nn_cls, _ = find_nearest_neighbors(k_array[k], dists, train_cls)
        e_rate = compute_error_rate(nn, test_cls, nn_cls)
        e_rates[k] = e_rate
        print('Error rate for k = {} is {}'.format(k_array[k], e_rate))

    plt.plot(k_array, e_rates)
    plt.ylabel('Error Rates')
    plt.xlabel('K')
    plt.show()

def get_dists(filename, train_imgs, test_imgs):
    # size of dists is 4000 x 10000
    if (os.path.isfile(filename)):
        dists = np.load(filename)
```

```python
        print('Precalculated distances loaded.')
    else:
        print('Calculating distances')
        dists = compute_dists(train_imgs, test_imgs)
        print('Saving distances')
        np.save(filename, dists)
        print('Distances saved')
    return dists


def run_KNN():

    # load CIFAR data
    train_imgs, train_cls, train_names = cifar_loader.load_training_data()
    test_imgs, test_cls, test_names = cifar_loader.load_test_data()
    class_names = np.array(cifar_loader.load_class_names())

    # get relevant class data
    labels_to_keep = np.sort(np.array([0, 1, 8, 9]))
    train_imgs, train_cls = split_sets(train_imgs, train_cls, labels_to_keep)
    test_imgs, test_cls = split_sets(test_imgs, test_cls, labels_to_keep)
    class_names = split_class_names(class_names, labels_to_keep)

    # part a
    view_examples(train_imgs, train_cls, labels_to_keep, class_names)

    # part b and c
    train_imgs_gray = rgb2gray_luminosity(train_imgs)
    test_imgs_gray = rgb2gray_luminosity(test_imgs)
    dists = get_dists('dists_lum_eucl.npy',train_imgs_gray, test_imgs_gray)



    k_array = np.array([1,2,5,10,20])
    plot_k_error_rates(k_array, dists, train_cls, test_cls)


    # part d
    plot_nearest_neighbors(labels_to_keep, dists, train_cls, test_cls, test_imgs,
        train_imgs, class_names)






if __name__ == '__main__':
     run_KNN()
```

where the *cifar_loader* file is included in the code. The assignment is run through the function *run_KNN*().

Part a: Random Image Visualization

I've chosen the classes

$$'[airplane',' automobile',' ship',' truck']$$

which correspond with labels

$$[0, 1, 8, 9]$$

The random image visualization is accomplished by the function $view\_examples()$ and is called on line 181, with the function definition on lines 28-42. Calling the code once returns the figure below

Figure 1: Random Image Examples 1

Proving that it is indeed random, calling it again returns

Figure 2: Random Image Examples 2

Part b and c: Applying KNN Algorithm for k = 1, 2, 5, 10, 20

The relevant code in $run\_KNN()$ is in lines 184-191. This function converts all the training and testing images to grayscale to compute one distance between two images as opposed to one for each channel (lines 184-185). I used two different grayscale conversions, one being the average of the rgb channels, and the other being a coefficient based conversion. This last algorithm emphasizes that the majority of color that we see is in the green channel. The code then computes a distance matrix on line 186, with the function $get\_dists()$, which looks for a saved distance matrix and loads it if it exists, and if not found, computes it. I the Euclidean Distance as a distance metric. The function $get\_dists()$ is on lines 153-164 and calls $compute\_dists()$, which is on lines 65-73. Once the distances have been calculated or loaded, it computes the error rates and plots them using $plot\_k\_error\_rates()$ (lines 140-151), $find\_nearest\_neighbors()$ (lines 75-84) and $compute\_error\_rate()$ (lines 86-111). If the predicted classes for the nearest neighbors are the same, it chooses the neighbor with the smallest average distance. The results for the different grayscale conversions are below

Average Grayscale and Euclidean Distance

Using the average grayscaling and Euclidean Distance gives the following output and graph

```
Error rate for k = 1 is 0.52975
Error rate for k = 2 is 0.52975
Error rate for k = 5 is 0.53325
Error rate for k = 10 is 0.5355
Error rate for k = 20 is 0.549
```

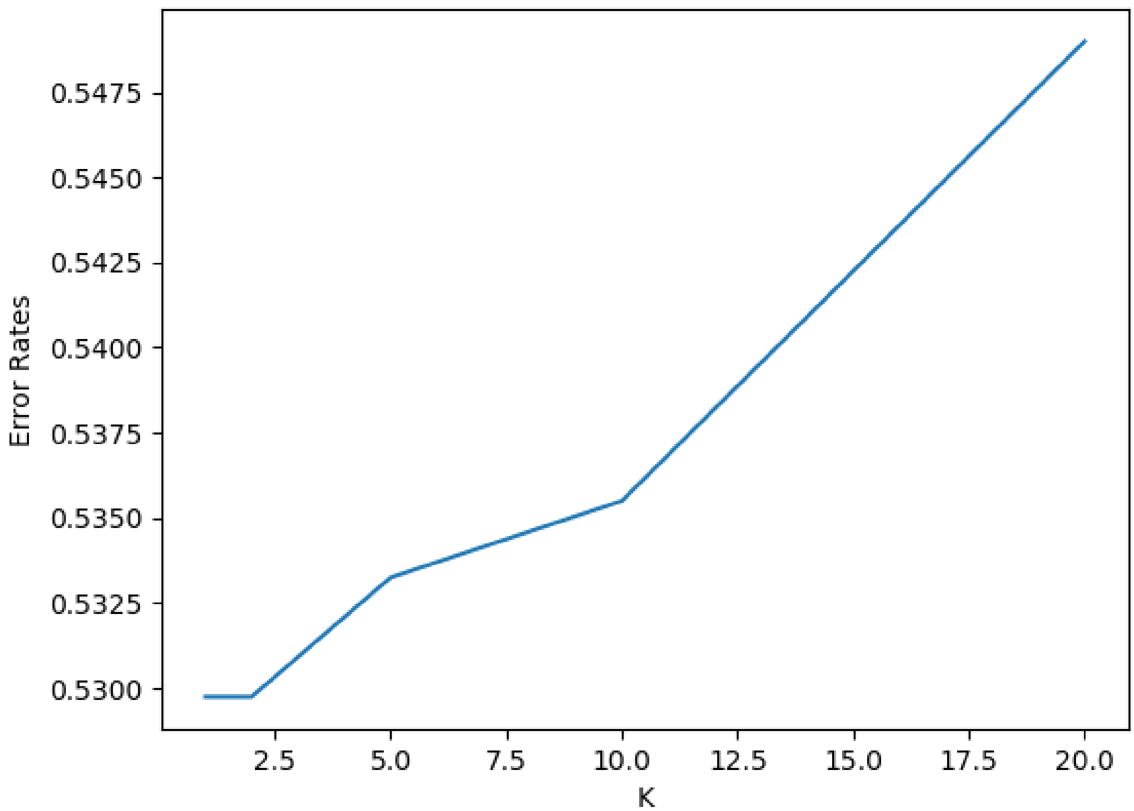

Figure 3: K vs Error Rate Using Average Grayscaling and Euclidean Distance

A more comprehensive sweep, using k's from 1-20 gives

```
Error rate for k = 1 is 0.52975
Error rate for k = 2 is 0.52975
Error rate for k = 3 is 0.52925
Error rate for k = 4 is 0.53025
Error rate for k = 5 is 0.53325
Error rate for k = 6 is 0.5305
Error rate for k = 7 is 0.5355
Error rate for k = 8 is 0.53675
Error rate for k = 9 is 0.53325
Error rate for k = 10 is 0.5355
```

```
Error rate for k = 11 is 0.537
Error rate for k = 12 is 0.53825
Error rate for k = 13 is 0.5405
Error rate for k = 14 is 0.54175
Error rate for k = 15 is 0.54175
Error rate for k = 16 is 0.54525
Error rate for k = 17 is 0.54825
Error rate for k = 18 is 0.54875
Error rate for k = 19 is 0.54825
Error rate for k = 20 is 0.549
```

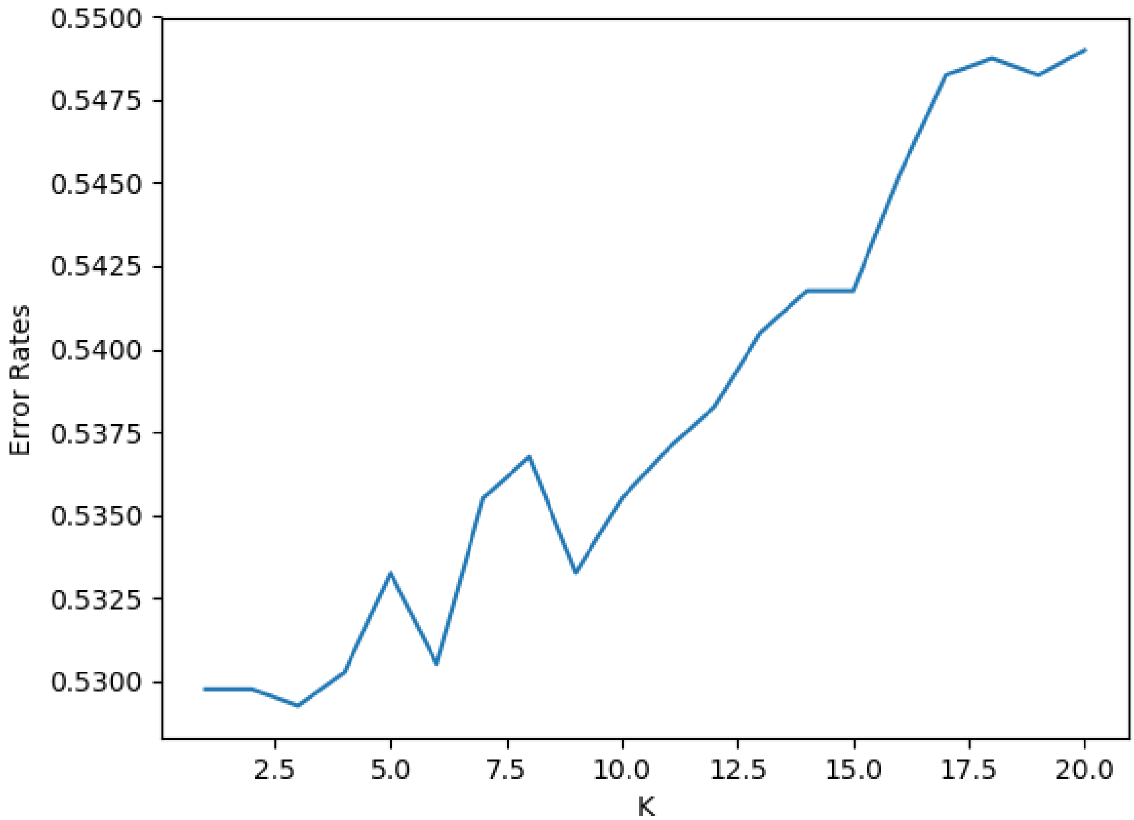

Figure 4: K vs Error Rate Using Average Grayscaling and Euclidean Distance

Which shows that the minimum error rate is again given by k = 3, and the error rate for k = 1 is 0.52975

Coefficient Grayscale and Euclidean Distance

Using the coefficient grayscaling and Euclidean Distance gives the following output and graph

```
Error rate for k = 1 is 0.53125
Error rate for k = 2 is 0.53125
Error rate for k = 5 is 0.54
Error rate for k = 10 is 0.54275
Error rate for k = 20 is 0.5475
```

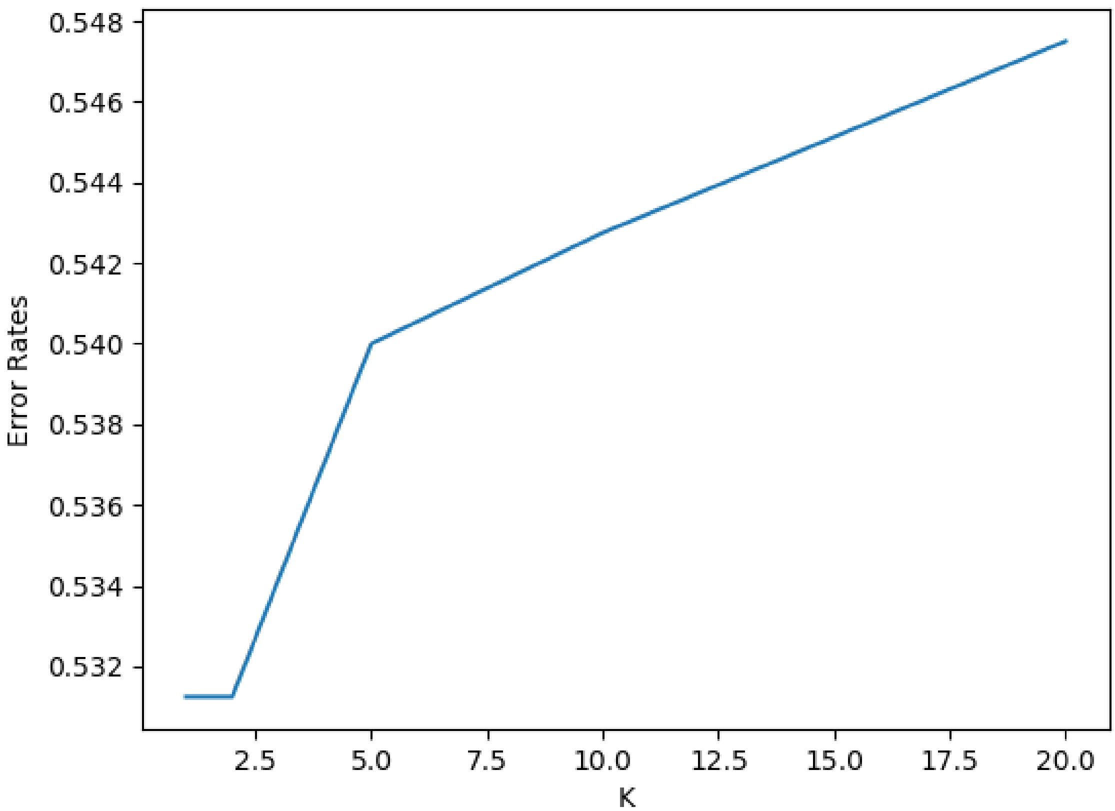

Figure 5: K vs Error Rate Using Coefficient Grayscaling and Euclidean Distance

A more comprehensive sweep, using k's from 1-20 gives

```
Error rate for k = 1 is 0.53125
Error rate for k = 2 is 0.53125
Error rate for k = 3 is 0.53875
Error rate for k = 4 is 0.54025
Error rate for k = 5 is 0.54
Error rate for k = 6 is 0.542
Error rate for k = 7 is 0.545
Error rate for k = 8 is 0.54075
Error rate for k = 9 is 0.54125
Error rate for k = 10 is 0.54275
```

```
Error rate for k = 11 is 0.54075
Error rate for k = 12 is 0.546
Error rate for k = 13 is 0.54575
Error rate for k = 14 is 0.54625
Error rate for k = 15 is 0.546
Error rate for k = 16 is 0.5465
Error rate for k = 17 is 0.54625
Error rate for k = 18 is 0.5485
Error rate for k = 19 is 0.54475
Error rate for k = 20 is 0.5475
```

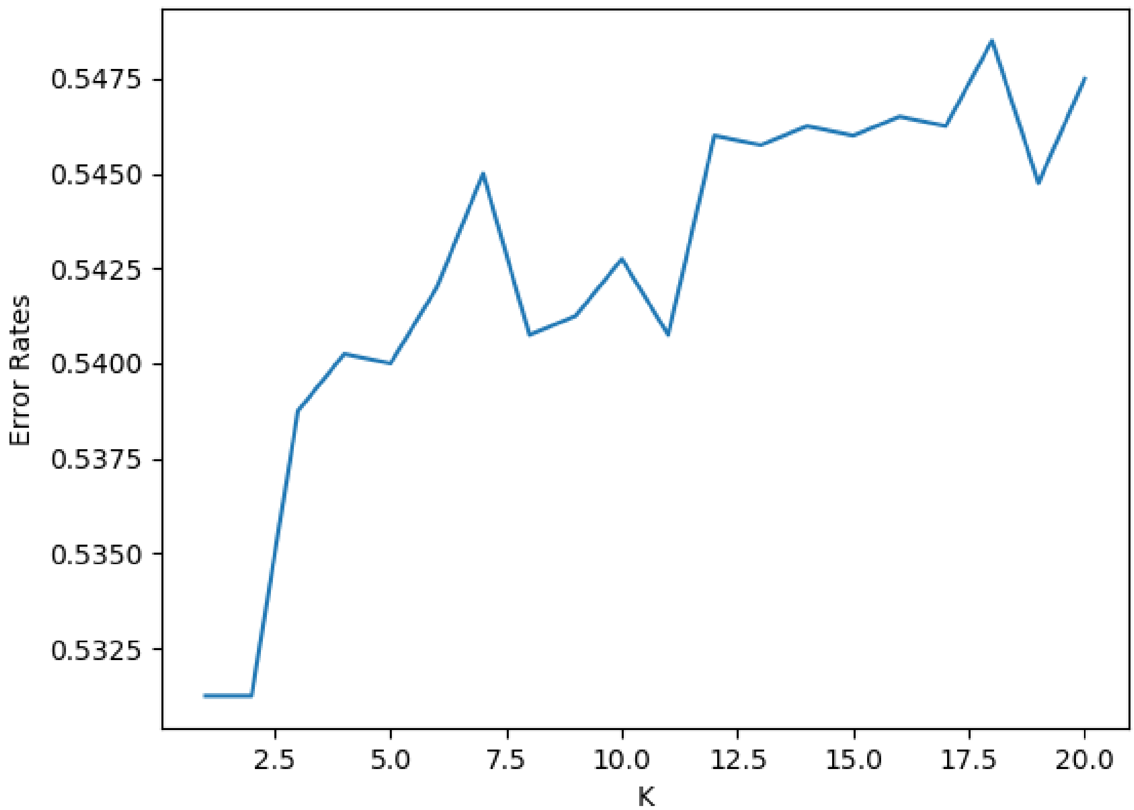

Figure 6: K vs Error Rate Using Coefficient Grayscaling and Euclidean Distance

Which shows that the minimum error rate is again given by k = 1,2, and the error rate for k = 1 is 0.53125.

In general, the error rate does not decrease with k, and it shouldn't since it is a dataset dependent variable.

1 Part d: Visualizing the 5 Nearest Neighbors

This part is accomplished with the function call on 196 (definition on lines 113-138). Two runs of the code gives the plots below which show the randomness of the image choice.

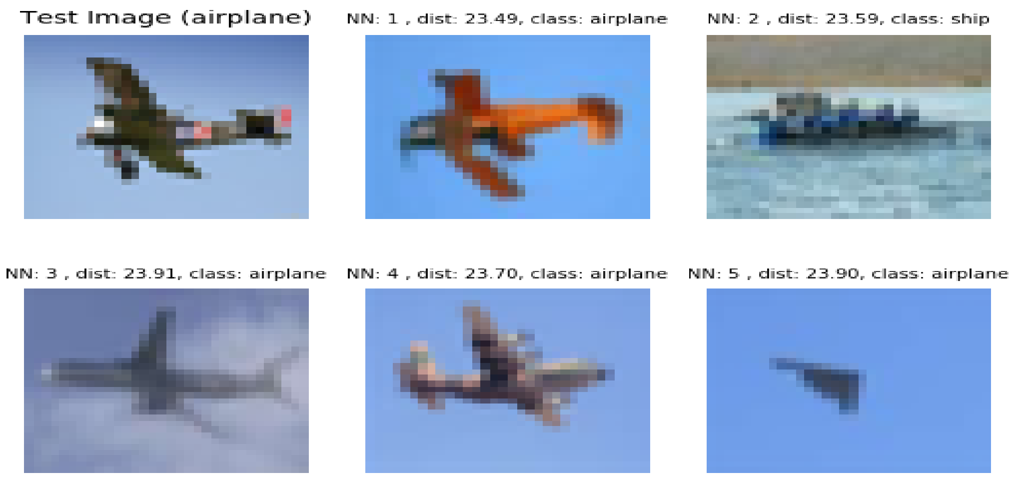

Figure 7: Random Airplane Image and it's 5 Nearest Neighbors

Figure 8: Random Airplane Image and it's 5 Nearest Neighbors

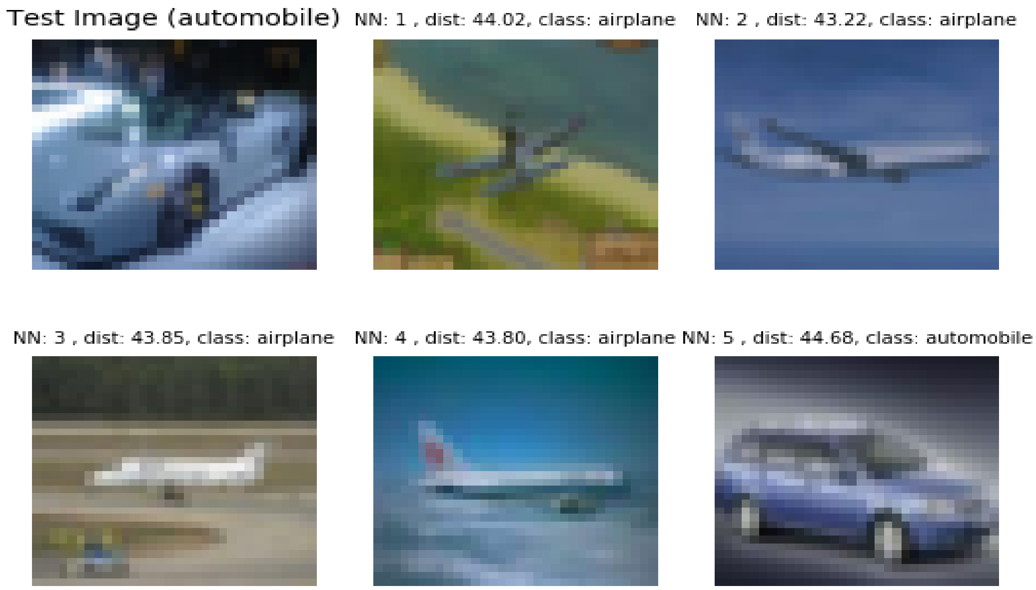

Figure 9: Random Automobile Image and it's 5 Nearest Neighbors

Figure 10: Random Automobile Image and it's 5 Nearest Neighbors

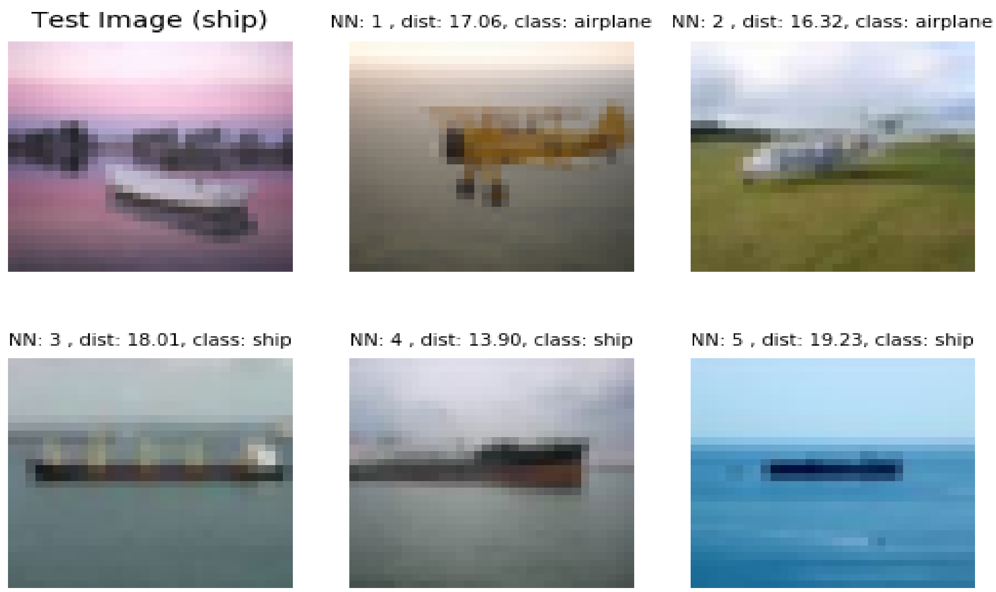

Figure 11: Random Ship Image and it's 5 Nearest Neighbors

Figure 12: Random Ship Image and it's 5 Nearest Neighbors

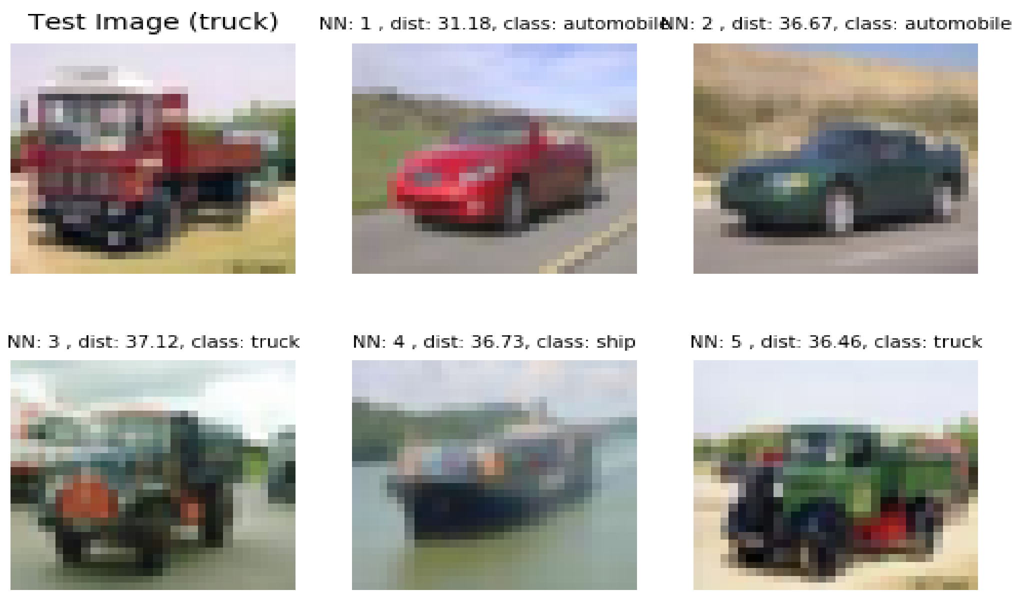

Figure 13: Random Truck Image and it's 5 Nearest Neighbors

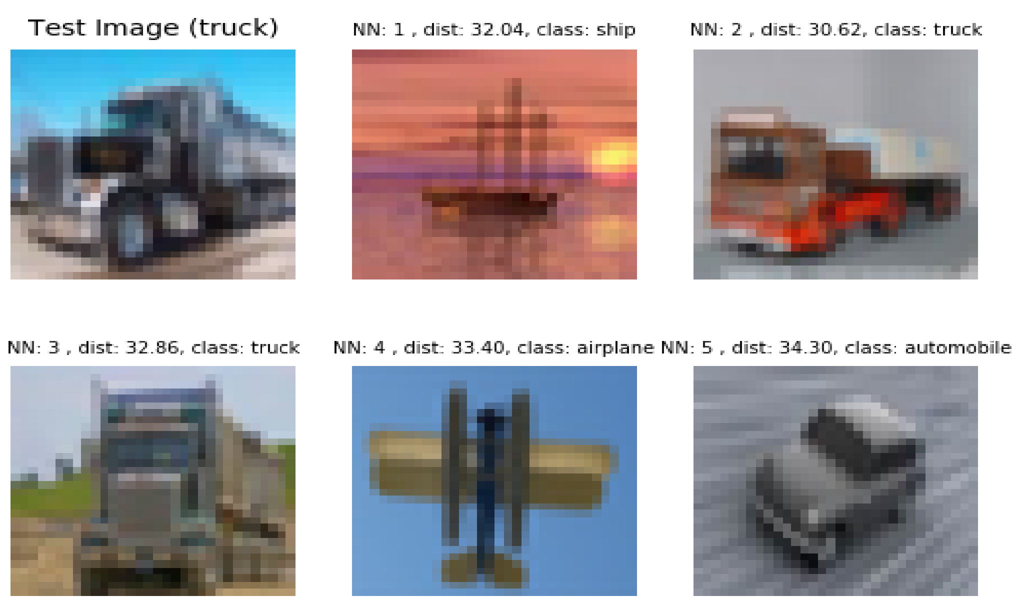

Figure 14: Random Truck Image and it's 5 Nearest Neighbors