k-Nearest Neighbor

In [1]:

```
###### IMPORTS ######
import os
import glob
from imutils import paths
import cv2
import numpy as np
import matplotlib.pyplot as plt
from knn_classifier import kNearestNeighbor

# This is a bit of magic to make matplotlib figures appear inline in the notebook
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (12, 9) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python modules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

In [2]:

```
###### GLOBAL VARIABLES ######
# Labels
BACKGROUND = 0
POTATO = 1
CARROT = 2
CAT_SAL = 3
CAT_BEEF = 4
BUN = 5
ARM = 6
KETCHUP = 7

# Background mask
BGD_MASK = cv2.imread('../preprocessing/bgd_mask.jpg', cv2.IMREAD_COLOR)

# Average background image
AVG_BGD_MASK = cv2.imread('../preprocessing/avg_background.jpg', cv2.IMREAD_COLOR)

# Test directories
BGD_TEST_DIR = glob.glob('../dataset3/res_crop/test/background/*.jpg')
POTATO_TEST_DIR = glob.glob('../dataset3/res_crop/test/potato/*.jpg')
CARROT_TEST_DIR = glob.glob('../dataset3/res_crop/test/carrots/*jpg')
CAT_SAL_TEST_DIR = glob.glob('../dataset3/res_crop/test/catfood_salmon/*.jpg')
CAT_BEEF_TEST_DIR = glob.glob('../dataset3/res_crop/test/catfood_beef/*.jpg')
BUN_TEST_DIR = glob.glob('../dataset3/res_crop/test/bun/*.jpg')
ARM_TEST_DIR = glob.glob('../dataset3/res_crop/test/arm/*.jpg')
KETCHUP_TEST_DIR = glob.glob('../dataset3/res_crop/test/ketchup/*.jpg')

# Train directories
BGD_TRAIN_DIR = glob.glob('../dataset3/res_crop/train/background/*.jpg')
POTATO_TRAIN_DIR = glob.glob('../dataset3/res_crop/train/potato/*.jpg')
CARROT_TRAIN_DIR = glob.glob('../dataset3/res_crop/train/carrots/*jpg')
CAT_SAL_TRAIN_DIR = glob.glob('../dataset3/res_crop/train/catfood_salmon/*.jpg')
CAT_BEEF_TRAIN_DIR = glob.glob('../dataset3/res_crop/train/catfood_beef/*.jpg')
BUN_TRAIN_DIR = glob.glob('../dataset3/res_crop/train/bun/*.jpg')
ARM_TRAIN_DIR = glob.glob('../dataset3/res_crop/train/arm/*.jpg')
KETCHUP_TRAIN_DIR = glob.glob('../dataset3/res_crop/train/ketchup/*.jpg')
```

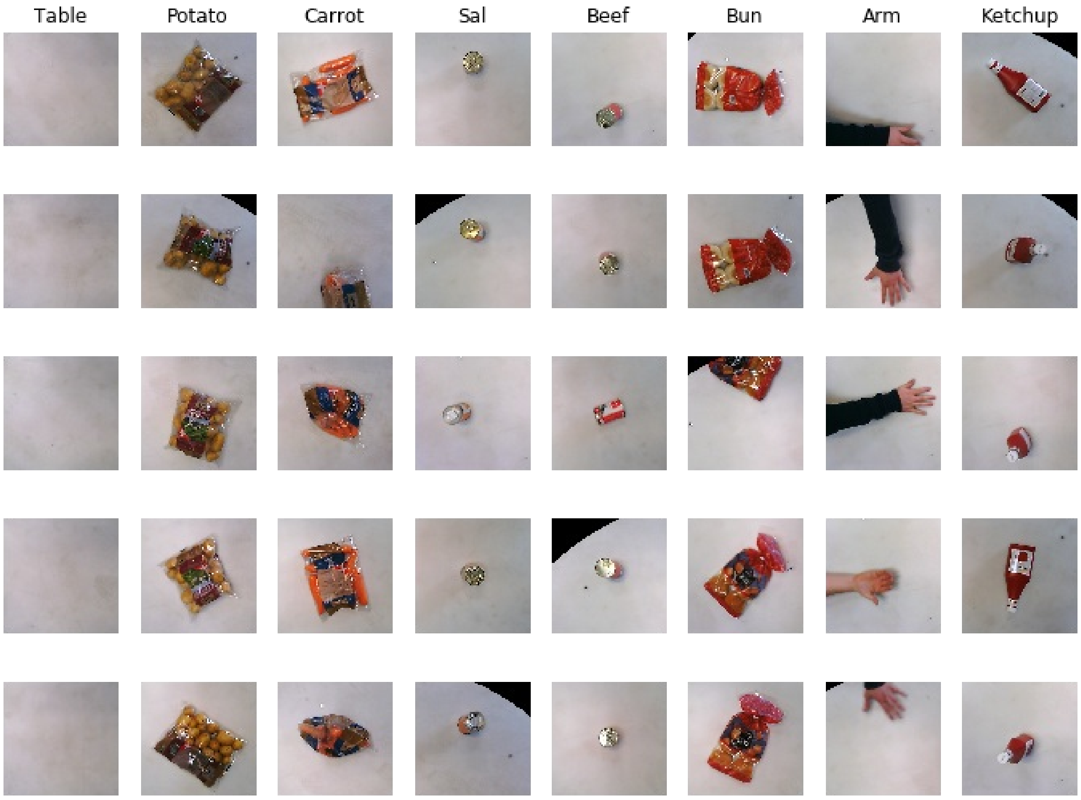
In [3]:

```
###### FUNCTIONS ######
def time_function(func, *args):
    """
    Call a function, func, with args and return the time, in [s],
    that it took to execute.
    """

    import time
    tic = time.time()
    func(*args)
    toc = time.time()

    return toc - tic

def extract_hist(image, bins=(8, 8, 8)):
    """
    Extracts a 3D color histogram from the hsv color space using
    the supplied number of bins per channel.
    """

    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist(images=[hsv],
                       channels=[0, 1, 2],
                       mask=None,
                       histSize=bins,
                       ranges=[0, 180, 0, 256, 0, 256])
    cv2.normalize(src=hist, dst=hist)
    return hist.flatten()

def image_to_feature_vector(image, size=(32, 32)):
    """
    resize the image to a fixed size, then flatten the image into
    a list of raw pixel intensities
    """

    return cv2.resize(image, size).flatten()
```

Cropped image with no filter

In [4]:

```
def convert_label_to_number(label):
    """ Converts label to number """

    result = None
    if label == 'bgd':
        result = 0
    elif label == 'potato':
        result = 1
    elif label == 'carrot':
        result = 2
    elif label == 'beef':
        result = 3
    elif label == 'sal':
        result = 4
    elif label == 'bun':
        result = 5
    elif label == 'arm':
        result = 6
    elif label == 'ketchup':
        result = 7
    return result

def import_data(X_hist, X_pixel, y, path, show_img):
    """ Imports data """

    image_paths = list(paths.list_images(path))

    if show_img is True:
        images = []

    for i, image_path in enumerate(image_paths):
        img = cv2.imread(image_path, cv2.IMREAD_COLOR)
        label = image_path.split(os.path.sep)[-1].split('_')[0]
        label = convert_label_to_number(label)

        #Pixel
        #Pixels
        pixel = image_to_feature_vector(img)
        X_pixel.append(pixel)

        # Histogram
        hist = extract_hist(img)
        X_hist.append(hist)

        y.append(label)

        if show_img is True:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            images.append(img)

    if show_img is True:
        return images

###### IMPORT TRAIN DATA ######
X_train_hist = []
X_train_pixel = []
y_train = []
path = 'cropped_images/train/'
import_data(X_train_hist, X_train_pixel, y_train, path, False)

###### IMPORT TEST DATA ######
X_test_hist = []
X_test_pixel = []
y_test = []
path = 'cropped_images/test/'
images = import_data(X_test_hist, X_test_pixel, y_test, path, True)
classes = ['Table','Potato','Carrot','Sal','Beef','Bun','Arm','Ketchup']
num_classes = len(classes)
samples_per_class = 5
plt.figure(figsize=(12, 9))
for y, cls in enumerate(classes):
    idxs = [i for i, label in enumerate(y_test) if label == y]
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(images[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls)
plt.show()
```

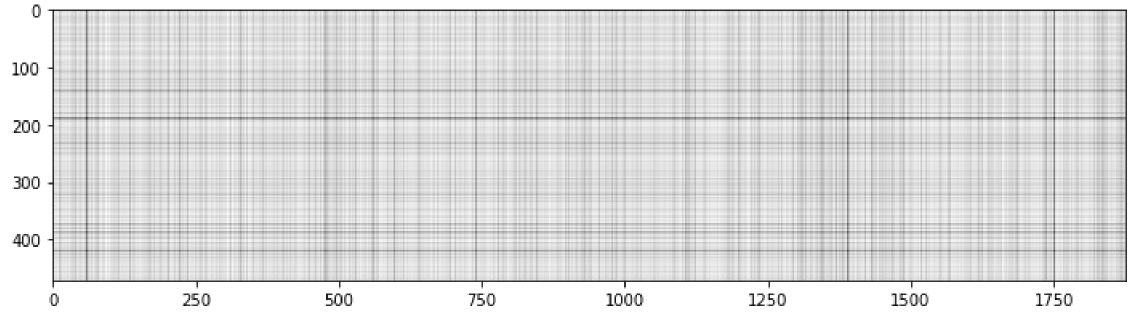


Information

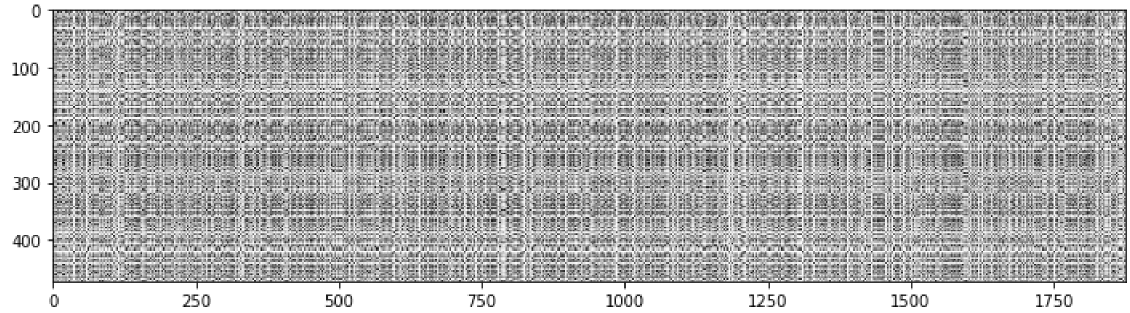
In [5]:

```
###### INFORMATION ######
# Pixel
print('\nPixel:')

X_train_pixel = np.array(X_train_pixel)
y_train = np.array(y_train)
X_test_pixel = np.array(X_test_pixel)
y_test = np.array(y_test)
print('Histogram train matrix: {:.2f}MB'.format(X_train_pixel.nbytes / (1024 * 1000.0)))
print('Histogram test matrix: {:.2f}MB'.format(X_test_pixel.nbytes / (1024 * 1000.0)))
print('Train data shape:', X_train_pixel.shape)
print('Tain labels shape:', y_train.shape)
print('Test data shape:', X_test_pixel.shape)
print('Test labels shape:', y_test.shape)

###### CREATE AND TRAIN CLASSIFIER ######
classifier = kNearestNeighbor()
classifier.train(X_train_pixel, y_train)

# Test implementation
dists = classifier.compute_distances_no_loops(X_test_pixel)
print('Distance shape:', dists.shape)
plt.figure(figsize=(12, 9))
plt.imshow(dists, interpolation='none')
plt.show()

# Time performance
time = time_function(classifier.compute_distances_no_loops, X_test_pixel)
print('Time performance:', time, 'seconds')

# Histogram
print('\nHISTOGRAM:')

X_train_hist = np.array(X_train_hist)
y_train = np.array(y_train)
X_test_hist = np.array(X_test_hist)
y_test = np.array(y_test)
print('Histogram train matrix: {:.2f}MB'.format(X_train_hist.nbytes / (1024 * 1000.0)))
print('Histogram test matrix: {:.2f}MB'.format(X_test_hist.nbytes / (1024 * 1000.0)))
print('Train data shape:', X_train_hist.shape)
print('Tain labels shape:', y_train.shape)
print('Test data shape:', X_test_hist.shape)
print('Test labels shape:', y_test.shape)

###### CREATE AND TRAIN CLASSIFIER ######
classifier = kNearestNeighbor()
classifier.train(X_train_hist, y_train)

# Test implementation
dists = classifier.compute_distances_no_loops(X_test_hist)
print('Distance shape:', dists.shape)
plt.figure(figsize=(12, 9))
plt.imshow(dists, interpolation='none')
plt.show()

# Time performance
time = time_function(classifier.compute_distances_no_loops, X_test_hist)
print('Time performance:', time, 'seconds')
```

```
Pixel:
Histogram train matrix: 5.62MB
Histogram test matrix: 1.42MB
Train data shape: (1875, 3072)
Tain labels shape: (1875,)
Test data shape: (472, 3072)
Test labels shape: (472,)
Distance shape: (472, 1875)
```



```
Time performance: 1.6034114360809326 seconds

HISTOGRAM:
Histogram train matrix: 3.75MB
Histogram test matrix: 0.94MB
Train data shape: (1875, 512)
Tain labels shape: (1875,)
Test data shape: (472, 512)
Test labels shape: (472,)
Distance shape: (472, 1875)
```



```
Time performance: 0.008638620376586914 seconds
```

Cross-validation

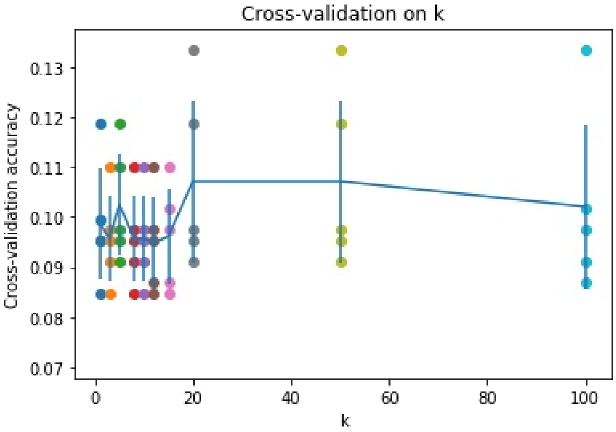
In [6]:

```python
def cross_val(k_choices, num_folds, X_train_folds, y_train_folds, shape):
    """ Performs cross validation """

    # A dictionary holding the accuracies to find the best value of k
    k_to_accuracies = {}

    # Perform k-fold cross validation to find the best value of k
    for k in k_choices:
        for j in range(num_folds):
            all_but_one_ind = [i for i in range(num_folds) if i != j]
            X_all_but_one = np.concatenate(X_train_folds[all_but_one_ind])
            y_all_but_one = np.concatenate(y_train_folds[all_but_one_ind])

            knn = kNearestNeighbor()
            knn.train(X_all_but_one, y_all_but_one)
            y_pred_k_f = knn.predict(X_train_folds[j], k)

            acc = float(sum(y_pred_k_f == y_train_folds[j])) / shape

            if k not in k_to_accuracies:
                k_to_accuracies[k] = []
            k_to_accuracies[k].append(acc)

    # Plot the raw observations
    for k in k_choices:
        accuracies = k_to_accuracies[k]
        plt.scatter([k] * len(accuracies), accuracies)

    # Plot the trend line with error bars that corresponde to standard deviation
    accuracies_mean = np.array([np.mean(v) for k, v in sorted(k_to_accuracies.items())])
    accuracies_std = np.array([np.std(v) for k, v in sorted(k_to_accuracies.items())])
    plt.errorbar(k_choices, accuracies_mean, yerr=accuracies_std)
    plt.title('Cross-validation on k')
    plt.xlabel('k')
    plt.ylabel('Cross-validation accuracy')
    plt.show()
```

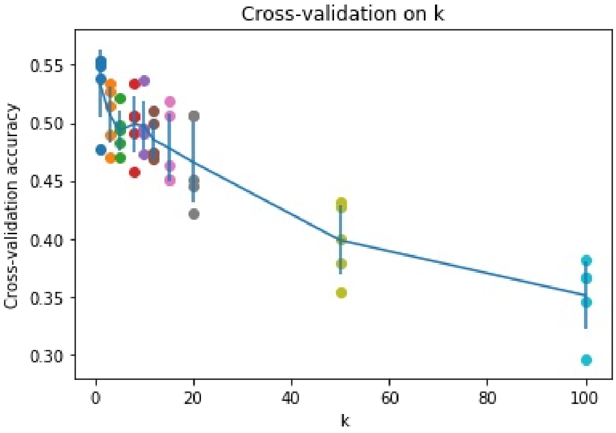
In [7]:

```
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

X_train_hist_folds = []
X_train_pixel_folds = []
y_train_folds = []

# Split training data into folds
X_train_hist_folds = np.array(np.array_split(X_train_hist, num_folds))
X_train_pixel_folds = np.array(np.array_split(X_train_pixel, num_folds))
y_train_folds = np.array(np.array_split(y_train, num_folds))

print('Cross-validation for pixel feature')
cross_val(k_choices, num_folds, X_train_pixel_folds, y_train_folds, X_test_pixel.shape[0])

print('Cross-validation for histogram feature:')
cross_val(k_choices, num_folds, X_train_hist_folds, y_train_folds, X_test_hist.shape[0])
```

```
Cross-validation for pixel feature
```



```
Cross-validation for histogram feature:
```



Best value of k

Based on the cross-validation results above, the best value for k is choosen. The classifier is retrained using all the training data and then tested on the test data.

In [8]:

```
###### PIXEL BY PIXEL ######
print('kNN with the best value of k on pixel feature:')
best_k = 1

classifier = kNearestNeighbor()
classifier.train(X_train_pixel, y_train)
y_test_pred = classifier.predict(X_test_pixel, k=best_k)

# Compute and display the accuracy
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / X_test_pixel.shape[0]
print('Got %d / %d correct' % (num_correct, X_test_pixel.shape[0]))
print('Accuracy: %f' % accuracy)

###### HISTOGRAM ######
print('\nkNN with the best value of k on histogram feature:')
best_k = 1

classifier = kNearestNeighbor()
classifier.train(X_train_hist, y_train)
y_test_pred = classifier.predict(X_test_hist, k=best_k)

# Compute and display the accuracy
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / X_test_hist.shape[0]
print('Got %d / %d correct' % (num_correct, X_test_hist.shape[0]))
print('Accuracy: %f' % accuracy)
```

```
kNN with the best value of k on pixel feature:
Got 66 / 472 correct
Accuracy: 0.139831

kNN with the best value of k on histogram feature:
Got 348 / 472 correct
Accuracy: 0.737288
```

Cropped images with filter

In [9]:

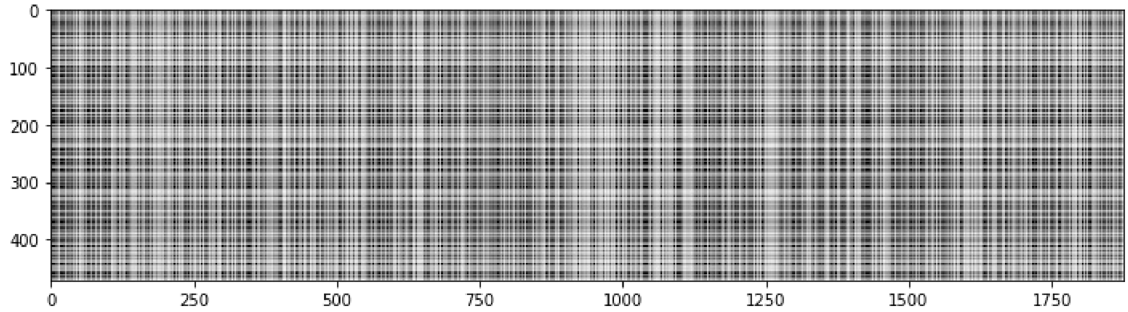
```
def convert_label_to_number(label):
    """ Converts label to number """

    result = None
    if label == 'bgd':
        result = 0
    elif label == 'potato':
        result = 1
    elif label == 'carrot':
        result = 2
    elif label == 'beef':
        result = 3
    elif label == 'sal':
        result = 4
    elif label == 'bun':
        result = 5
    elif label == 'arm':
        result = 6
    elif label == 'ketchup':
        result = 7
    return result

def import_data(X_hist, X_pixel, y, path, show_img):
    """ Imports data """

    lower = (0, 65, 0)
    upper = (179, 255, 255)

    image_paths = list(paths.list_images(path))

    if show_img is True:
        images = []

    for i, image_path in enumerate(image_paths):
        image = cv2.imread(image_path, cv2.IMREAD_COLOR)
        label = image_path.split(os.path.sep)[-1].split('_')[0]
        label = convert_label_to_number(label)
        label = convert_label_to_number(label)

        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(src=hsv, lowerb=lower, upperb=upper)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (30, 30))
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
        img = cv2.bitwise_and(image, image, mask=mask)
        #img = cv2.bitwise_and(img, BGD_MASK)

        # Pixel
        pixel = image_to_feature_vector(img)
        X_pixel.append(pixel)

        # Histogram
        hist = extract_hist(img)
        X_hist.append(hist)

        y.append(label)

        if show_img is True:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            images.append(img)

    if show_img is True:
        return images

###### IMPORT TRAIN DATA ######
X_train_hist = []
X_train_pixel = []
y_train = []

path = 'cropped_images/train/'
import_data(X_train_hist, X_train_pixel, y_train, path, False)

###### IMPORT TEST DATA ######
X_test_hist = []
X_test_pixel = []
y_test = []

path = 'cropped_images/test/'
images = import_data(X_test_hist, X_test_pixel, y_test, path, True)

classes = ['Table','Potato','Carrot','Sal','Beef','Bun','Arm','Ketchup']
num_classes = len(classes)
samples_per_class = 5
plt.figure(figsize=(12, 9))
for y, cls in enumerate(classes):
    idxs = [i for i, label in enumerate(y_test) if label == y]
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(images[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls)
plt.show()
```

Information

In [10]:

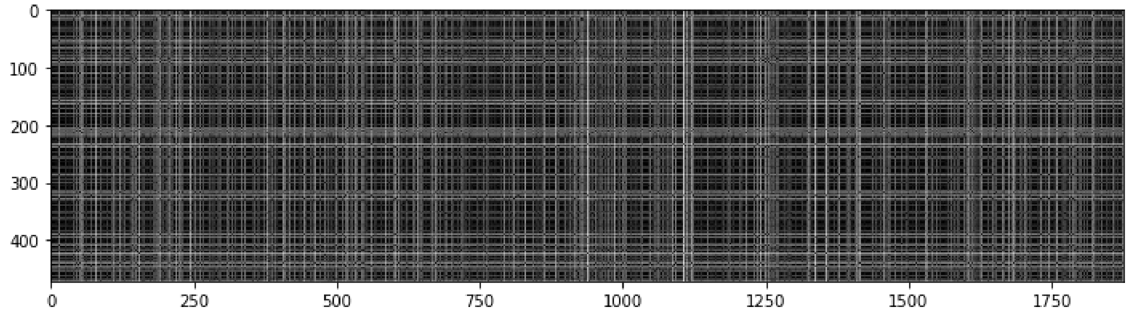
```
###### INFORMATION ######
# Pixel
print('\nPixel:')

X_train_pixel = np.array(X_train_pixel)
y_train = np.array(y_train)
X_test_pixel = np.array(X_test_pixel)
y_test = np.array(y_test)
print('Histogram train matrix: {:.2f}MB'.format(X_train_pixel.nbytes / (1024 * 1000.0)))
print('Histogram test matrix: {:.2f}MB'.format(X_test_pixel.nbytes / (1024 * 1000.0)))
print('Train data shape:', X_train_pixel.shape)
print('Tain labels shape:', y_train.shape)
print('Test data shape:', X_test_pixel.shape)
print('Test labels shape:', y_test.shape)

###### CREATE AND TRAIN CLASSIFIER ######
classifier = kNearestNeighbor()
classifier.train(X_train_pixel, y_train)

# Test implementation
dists = classifier.compute_distances_no_loops(X_test_pixel)
print('Distance shape:', dists.shape)
plt.figure(figsize=(12, 9))
plt.imshow(dists, interpolation='none')
plt.show()

# Time performance
time = time_function(classifier.compute_distances_no_loops, X_test_pixel)
print('Time performance:', time, 'seconds')

# Histogram
print('\nHISTOGRAM:')

X_train_hist = np.array(X_train_hist)
y_train = np.array(y_train)
X_test_hist = np.array(X_test_hist)
y_test = np.array(y_test)
print('Histogram train matrix: {:.2f}MB'.format(X_train_hist.nbytes / (1024 * 1000.0)))
print('Histogram test matrix: {:.2f}MB'.format(X_test_hist.nbytes / (1024 * 1000.0)))
print('Train data shape:', X_train_hist.shape)
print('Tain labels shape:', y_train.shape)
print('Test data shape:', X_test_hist.shape)
print('Test labels shape:', y_test.shape)

###### CREATE AND TRAIN CLASSIFIER ######
classifier = kNearestNeighbor()
classifier.train(X_train_hist, y_train)

# Test implementation
dists = classifier.compute_distances_no_loops(X_test_hist)
print('Distance shape:', dists.shape)
plt.figure(figsize=(12, 9))
plt.imshow(dists, interpolation='none')
plt.show()

# Time performance
time = time_function(classifier.compute_distances_no_loops, X_test_hist)
print('Time performance:', time, 'seconds')
```

```
Pixel:
Histogram train matrix: 5.62MB
Histogram test matrix: 1.42MB
Train data shape: (1875, 3072)
Tain labels shape: (1875,)
Test data shape: (472, 3072)
Test labels shape: (472,)
Distance shape: (472, 1875)
```



```
Time performance: 1.763817548751831 seconds
```

```
HISTOGRAM:
Histogram train matrix: 3.75MB
Histogram test matrix: 0.94MB
Train data shape: (1875, 512)
Tain labels shape: (1875,)
Test data shape: (472, 512)
Test labels shape: (472,)
Distance shape: (472, 1875)
```



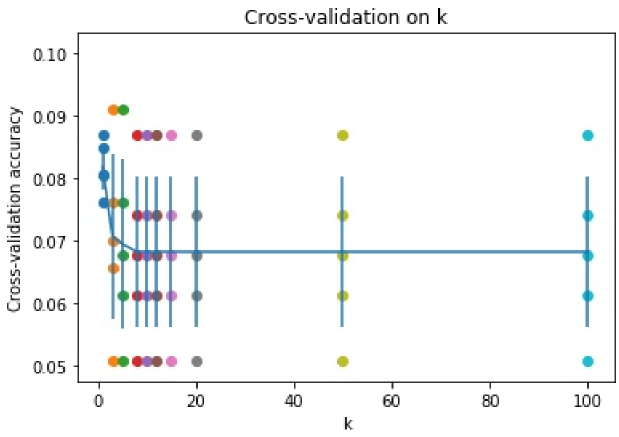
```
Time performance: 0.010046958923339844 seconds
```

Cross-validation

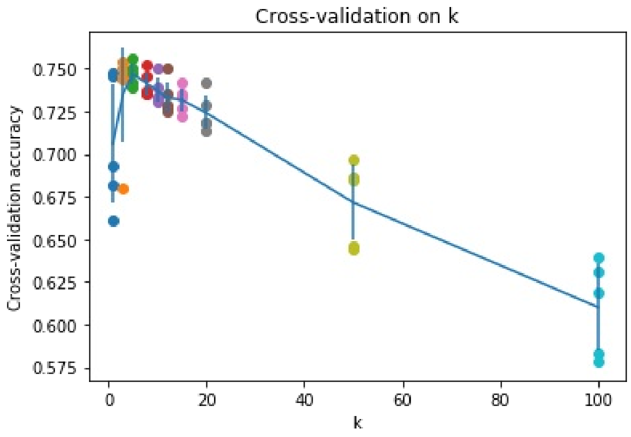
In [11]:

```
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

X_train_hist_folds = []
X_train_pixel_folds = []
y_train_folds = []

# Split training data into folds
X_train_hist_folds = np.array(np.array_split(X_train_hist, num_folds))
X_train_pixel_folds = np.array(np.array_split(X_train_pixel, num_folds))
y_train_folds = np.array(np.array_split(y_train, num_folds))

print('Cross-validation for pixel feature')
cross_val(k_choices, num_folds, X_train_pixel_folds, y_train_folds, X_test_pixel.shape[0])

print('Cross-validation for histogram feature:')
cross_val(k_choices, num_folds, X_train_hist_folds, y_train_folds, X_test_hist.shape[0])
```

```
Cross-validation for pixel feature
```



```
Cross-validation for histogram feature:
```



Best value of k

Based on the cross-validation results above, the best value for k is choosen. The classifier is retrained using all the training data and then tested on the test data.

In [12]:

```
###### PIXEL BY PIXEL ######
print('kNN with the best value of k on pixel feature:')
best_k = 3

classifier = kNearestNeighbor()
classifier.train(X_train_pixel, y_train)
y_test_pred = classifier.predict(X_test_pixel, k=best_k)

# Compute and display the accuracy
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / X_test_pixel.shape[0]
print('Got %d / %d correct' % (num_correct, X_test_pixel.shape[0]))
print('Accuracy: %f' % accuracy)

###### HISTOGRAM ######
print('\nkNN with the best value of k on histogram feature:')
best_k = 3

classifier = kNearestNeighbor()
classifier.train(X_train_hist, y_train)
y_test_pred = classifier.predict(X_test_hist, k=best_k)

# Compute and display the accuracy
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / X_test_hist.shape[0]
print('Got %d / %d correct' % (num_correct, X_test_hist.shape[0]))
print('Accuracy: %f' % accuracy)
```

```
kNN with the best value of k on pixel feature:
Got 41 / 472 correct
Accuracy: 0.086864

kNN with the best value of k on histogram feature:
Got 450 / 472 correct
Accuracy: 0.953390
```