[CS 4365/5354] Computer Vision - Lab 4 Report

Jose Perez

November 16, 2016

Contents

1. Lab 4

1.1 Introduction

Problem Clustering is the task of grouping a set of objects such that objects in the same group (cluster) are more similar to each other than those in other clusters. In the same way that sorting isn't an algorithm but a task, clustering is also not an algorithm but a task. Clustering in the context of computer vision has to do with grouping features from images such as pixel intensities and using them for applications such as image classification. In this laboratory I explore image clustering and classification.

There are several datasets used in computer vision for image classification purposes. This lab uses the CIFAR-10 and MNIST datasets.

The problems which this lab explores are:

Pixel Clustering
Provided an image represent it as best as possible using a limited amount of colors. The amount of colors is provided by the user.

MNIST Clustering w/Pixel Intensities Cluster the MNIST dataset using pixel intensities as features.

CIFAR-10 Clustering w/Color Histograms Cluster the MNIST dataset using color histograms as features.

CIFAR-10 Clustering w/Histograms of Oriented Gradients Cluster the MNIST dataset using histograms of oriented gradients as features.

MNIST Classification Classify the MNIST dataset using an algorithm and feature of choice.

CIFAR-10 Classification Classify the CIFAR-10 dataset using an algorithm and feature of choice.

1.2 Proposed Solution Design and Implementation

1.2.1 User Interface

This lab was created and tested using iPython and so it is recommended to use it for running this lab.

Modules I separated my code as follows:

lab4_problem1.py
: Given an image and a set amount of colors, clusters the colors in the image and recreates so it uses a limited amount of colors.

lab4_features.py
: Performs clustering and classification on a specified dataset using specified features. Allows the usage of K-Means clustering, Affinity Propagation clustering, Mini-Batch K-Means clustering, Mean Shift clustering, Multilayer Perceptrons, Convolutional Neural Networks, Linear Support Vector Machines, and 9-Poly Support Vector Machines. These classification algorithms along with the features provided in the library below are evaluated for accuracy.

lib_features.py
: Provides the implementation of features such as pixel intensities, color histograms, histograms of oriented gradients, and daisy features. This library is used for the classification and clustering of the datasets.

lib_cifar.py
: Library to load the CIFAR-10 dataset.

lib_mnist.py
: Library to load the MNIST dataset.

1.2.2 Implementation

Pixel Clustering
: Provided an image I and number of colors k.

  I is flattened and k-means clustering from the scikit-learn package is applied using the number of colors k. K-means gives the centroids and labels of the

flattened image. The colors of the image are replaced with the color of the centroid who is closest and the image is then shown with the new limited amount of colors.

Dataset Clustering and Classification Provided a dataset, classification algorithm, and feature set.

The dataset is split into training and validation sets. For MNIST these are provided and for CIFAR-10 75% of all 5 batches are used for training and the rest for validation.

The training set and validation set get transformed into a feature set based on the feature chosen. The feature library is used to perform this operation in the entire dataset. The feature set is then normalized for better performance (http://neerajkumar.org/writings/svm/).

The training feature set is then given to the chosen classification algorithm for training. If the classification involves clustering then information about the clusters is shown.

After training the classification algorithm the validation feature set is given to the algorithm for labeling. The classifier then labels each image in the validation set.

The accuracy of the classifier is evaluated by comparing the label given by the classifier with the ground truth provided by the dataset itself. $accuracy = imageslabeledcorrectly/totalnumberofimages$

The program then gives the accuracy as well as the running time of the program.

1.3 Experimental Results

The computer used to run these experiments is a Dell Inspiron 13 which has an Intel i7 Core and 12GB of memory.

These experiments were run in the Anaconda2 python environment using the Spyder IDE in an iPython console.

The timing (named duration below) of these experiments was taken using the default_timer of Python's timeit library.

As stated above, the datasets used were the CIFAR-10 (https://www.cs.toronto.edu/ kriz/-cifar.html) and MNIST (http://yann.lecun.com/exdb/mnist/) datasets.

For the problem of creating an image with a limited amount of colors the following images were used.

Figure 1.1: flower.jpg

Figure 1.2: landscape_s.jpg

The results of the color classification were as follows:

Figure 1.3: Classification of flower.jpg using k=1, 2, .., 12

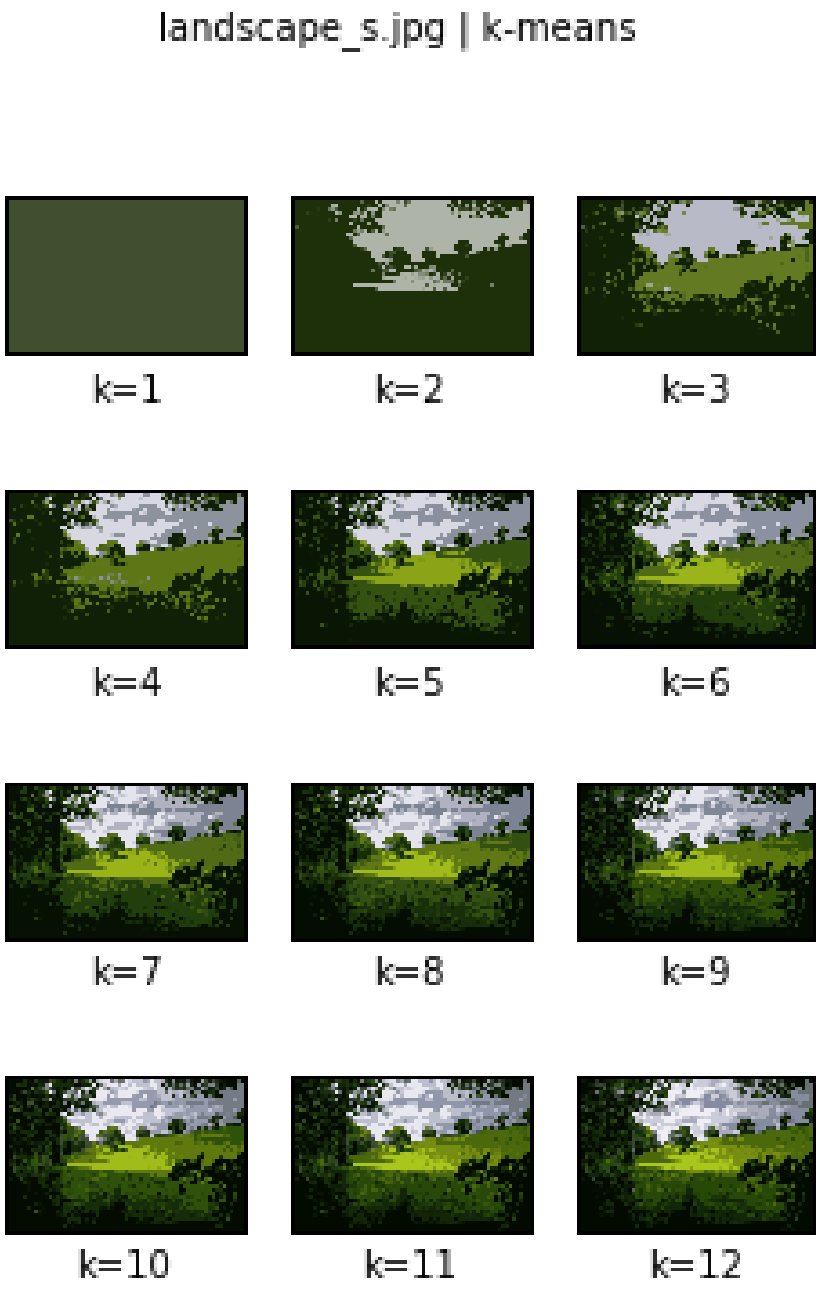

Figure 1.4: Classification of landscape_s.jpg using k=1, 2, .., 12

Of particular interest are the cluster centers when running k-means on the MNIST dataset using pixels.

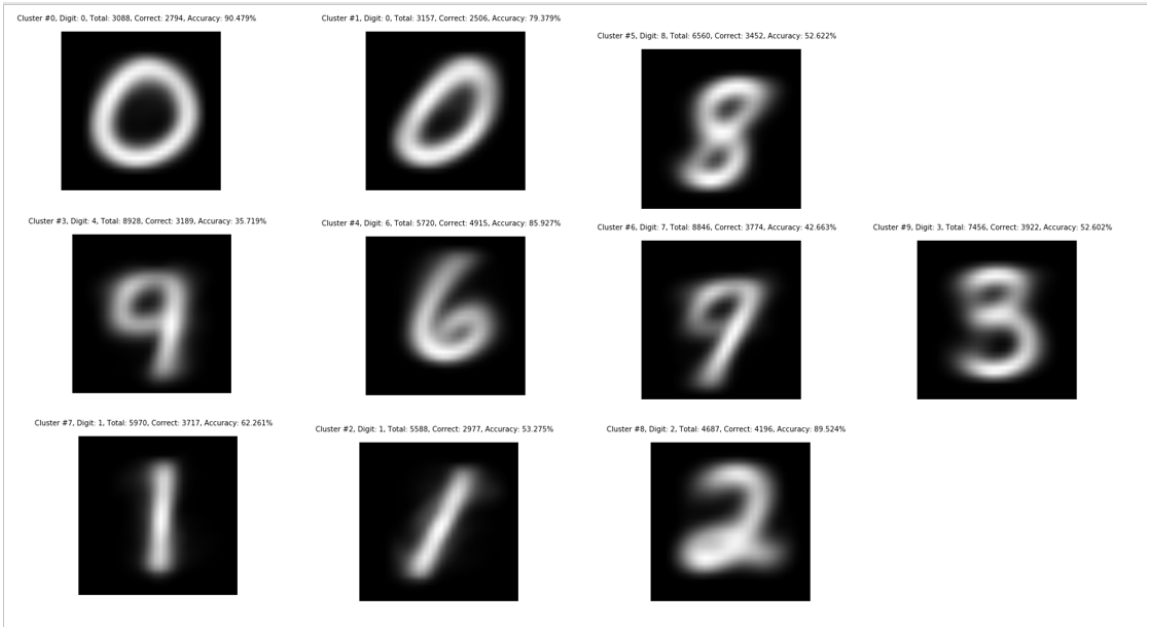

Figure 1.5: Cluster centers of MNIST using k=10

The results of the classification were as follows:

Table 1.1: Classification Results

Dataset	Algorithm	Feature	Running Time	Accuracy
CIFAR-10	512 neuron CNN	HOG	38 min	54.176%
CIFAR-10	MLP2	HOG	38.291s	51.808%
CIFAR-10	MLP	HOG	13.470s	50.16%
MNIST	5000 neuron MLP2	Pixel Int.	19 min	97.87%
MNIST	8000 neuron MLP2	Pixel Int.	29 min	97.78%
MNIST	MLP	Daisy	13 min	96.9%
MNIST	9-poly SVM	Pixel Int.	20 min	96.57%
MNIST	Linear SVM	Pixel Int.	9 min	90.89%

1.4 Conclusion

The best results for CIFAR-10 were acquired using a CNN with Histograms of Oriented Gradients. At first I was using only 1 HOG but then I decided to divide the image into 4 regions and take the HOG of each subregion and that performed better.

The best results for MNIST were acquired using 5000 neuron multilayer perceptrons using pixel intensities. Interestingly enough, the 8000 neuron attempt did worse.

Although my classification attempts are not close to the current state of the art they provided me a lot of insight into image clustering/classification and machine learning.

I created the code in a way that was easily extensible so I can try it with other datasets and more features in the future.

A. Source Code

lab4_problem1.py

```
# -*- coding: utf-8 -*-
"""
Course: CS 4365/5354 [Computer Vision]
Author: Jose Perez [ID: 80473954]
Assignment: Lab 4
Instructor: Olac Fuentes
"""
from timeit import default_timer as timer
from PIL import Image
import numpy as np
import pylab as plt
from math import ceil
from sklearn.cluster import KMeans

# =============== Variables
filename = "landscape_s.jpg"
k_min = 1
k_max = 12
k_step = 1

# =============== Constants
DECIMALS = '{:.3f}' # Set the decimals for the timer results
IMAGE_COLUMNS = 3.0 # (Floating pt) Number of images to show per row in
   results

# ============================== Load image
im_original = np.array(Image.open(filename), dtype=np.float64)

# Flatten for k-means
(w, h, c) = im_original.shape
im_flat = np.reshape(im_original, (w * h, c))

# Prepare figure
figure = plt.figure(0, figsize=(4, 7))
figure.suptitle(filename + " | k-means" )
nrows = ceil(k_max / k_step / IMAGE_COLUMNS)

start_time = timer()
for k in range(k_min, k_max+k_step, k_step):
```

```
    temp = im_flat.copy()
    # Run K-Means
    start_time = timer()
    k_means = KMeans(n_clusters = k).fit(temp)
    #print "K-Means took ", time_format.format(timer() - start_time) , "s
       for k =", k

    start_time = timer()
    for cluster in range(k):
        temp[k_means.labels_ == cluster] = k_means.cluster_centers_[cluster
           ]

    #print "Replacing took ", time_format.format(timer() - start_time), "s"

    subplot = figure.add_subplot(nrows+1, IMAGE_COLUMNS, k / k_step)
    subplot.set_xticks(())
    subplot.set_yticks(())
    subplot.set_xlabel("k="+str(k))
    subplot.imshow(np.reshape(temp, (w, h, c)).astype(np.uint8))

plt.show()

print "Took ", DECIMALS.format(timer() - start_time) , "s"
```

lab4_features.py

```
# -*- coding: utf-8 -*-
"""
Course: CS 4365/5354 [Computer Vision]
Author: Jose Perez [ID: 80473954]
Assignment: Lab 4
Instructor: Olac Fuentes
"""
from timeit import default_timer as timer
import numpy as np
import pylab as plt
from math import ceil
from sklearn.cluster import KMeans
from sklearn.cluster import MiniBatchKMeans
from sklearn.cluster import AffinityPropagation
from sklearn.cluster import MeanShift
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# Requires scikit-learn 0.18 or above
from sklearn.neural_network import MLPClassifier

# Deep neural networks
#   Can be installed with
#       pip install scikit-neuralnetwork
```

```
#   Requires Theano
#       http://deeplearning.net/software/theano/install_windows.html#
   install-windows
from sknn.mlp import Classifier as MLPClassifier2
from sknn.mlp import Layer
from sknn.mlp import Convolution
from sknn.mlp import Native

# https://github.com/Lasagne/Lasagne
# Deep neural networks
import lasagne.layers as lasagne_l
import lasagne.nonlinearities as lasagne_nl

from lib_mnist import load_mnist
from lib_cifar import load_cifar
from lib_features import pixel_intensity
from lib_features import color_histogram
from lib_features import gradient_histogram
from lib_features import daisy_feature

import warnings
warnings.filterwarnings("ignore")

#
   ================================================================================

#
   ================================================================================

#
   ================================================================================

#
   ================================================================================

#
   ================================================================================

# ---------------------- Global Parameters
# Show all the images in each cluster?
# Only for clustering algorithms
SHOW_CLUSTER_IMAGES = False

# If showing images, show only the centroids?
SHOW_CENTROIDS_ONLY = False

# (Floating pt) Number of images to show per row in results
IMAGE_COLUMNS = 25.0

# Print the accuracy of each cluster?
# Not recommended for AffinityProp and MeanShift
```

```
PRINT_CLUSTER_INFO = False

# Number of K clusters (Only for KMeans and MiniBatchKMeans)
CLUSTERS = 10
# {MNIST, CIFAR10}
CURRENT_DATASET = "MNIST"

# Number of CIFAR-10 batches to use (Max: 5)
CIFAR_BATCHES = 5

# Number of bars in the histogram features (if used)
HISTOGRAMS = 16

# {PIXEL, COLORHIST, HOG, DAISY}
FEATURE = "PIXEL"

# Clustering: {KMEANS, AFFINITYPROP, MINIBATCHKMEANS, MEANSHIFT}
# Neural Networks: {MLP, MLP2, CNN}
# SVM: {LINEARSVC, SVC9}
ALGORITHM = "CNN"

# Required for SVMs
FEATURE_SCALING = True

# Number of units in neural networks
NEURONS = 100

# Epochs to train CNN
EPOCHS = 10

# Should we deskew the images if possible?
PERFORM_DESKEW = False

# Set the decimals for the timer results
DECIMALS = '{:.3f}'

# Uses small datasets for debugging
DEBUGGING = False

# To prevent overlapping figures
CURRENT_FIGURE = 0
#
   ================================================================================

#
   ================================================================================

#
   ================================================================================

#
```

```
    ================================================================================

#
    ================================================================================

# -------------------- Evaluates the accuracy of clustering classifiers
# -------------------- Needed for majority labeling of k-means
def evaluate_clustering(dataset, dataset_flat, predict_lbls, true_lbls,
    cluster_centers, label_names=None):
    global SHOW_CLUSTER_IMAGES
    global SHOW_CENTROIDS_ONLY
    global IMAGE_COLUMNS
    global PRINT_CLUSTER_INFO

    global CLUSTERS
    global CURRENT_DATASET
    global ALGORITHM

    global CURRENT_FIGURE
    global DECIMALS

    if CURRENT_DATASET == "MNIST":
        (overall_total, w, h) = dataset.shape
    else:
        (overall_total, w, h, c) = dataset.shape

    print "Checking the accuracy of ",ALGORITHM," using {0} images".format(
        overall_total)
    # Keep track of how many we get right
    overall_accurate = 0

    for cluster_index in range(CLUSTERS):
        # Get the current cluster
        cluster = dataset_flat[predict_lbls == cluster_index]
        cluster_total = cluster.shape[0]

        # Get the labels for the current cluster
        labels = true_lbls[predict_lbls == cluster_index]

        # Check if the cluster is empty
        if cluster_total == 0:
            if PRINT_CLUSTER_INFO:
                print "Cluster {0} is empty.".format(cluster_index)
            continue

        # Select the majority as the cluster label
        majority = np.argmax(np.bincount(labels.flatten()))

        # How many did we cluster correctly?
        cluster_total_accurate = labels[labels == majority].shape[0]
```

```
    # Keep track of our overall accuracy
    overall_accurate += cluster_total_accurate

    # Figure out the accuracy
    accuracy = DECIMALS.format(cluster_total_accurate / float(
       cluster_total) * 100)

    if CURRENT_DATASET == "MNIST":
        title = "Cluster #{0}, Digit: {1}, {2}/{3} correct, Accuracy:
           {4}%".format(cluster_index, majority, cluster_total_accurate
           , cluster_total, accuracy)
    else:
        title = "Cluster #{0}, {1}, {2}/{3} correct, Accuracy: {4}%".
           format(cluster_index, label_names[majority],
           cluster_total_accurate,  cluster_total, accuracy)

    # Can only show centers for clustering algorithms
    if SHOW_CLUSTER_IMAGES:
        figure = plt.figure(CURRENT_FIGURE)
        figure.suptitle(title)

        CURRENT_FIGURE += 1
        offset = 1

        nrows = ceil(cluster_total / IMAGE_COLUMNS)

        if FEATURE == "PIXEL":
            if SHOW_CENTROIDS_ONLY:
                subplot = figure.add_subplot(1, 1, 1)
            else: # Show centroid in the middle of the first row
                subplot = figure.add_subplot(nrows+1, IMAGE_COLUMNS,
                   ceil(IMAGE_COLUMNS / 2.0))

            subplot.set_xticks(())
            subplot.set_yticks(())
            if CURRENT_DATASET == "MNIST":
                subplot.imshow(np.reshape(cluster_centers[cluster_index
                   ], (w, h)))
            else:
                subplot.imshow(np.reshape(cluster_centers[cluster_index
                   ], (w, h, c)))

            offset = 6
            nrows += 1

        if not SHOW_CENTROIDS_ONLY:
            # Show all the images in the cluster depending on the
               number of columns selected
            for image_index in range(cluster_total):
                subplot = figure.add_subplot(nrows+1, IMAGE_COLUMNS,
                   image_index+offset)
```

```
                    subplot.set_xticks(())
                    subplot.set_yticks(())

                    if CURRENT_DATASET == "MNIST":
                        subplot.imshow(np.reshape(dataset[image_index], (w,
                            h)))
                    else:
                        subplot.imshow(np.reshape(dataset[image_index], (w,
                            h, c)))

                plt.show()
            elif PRINT_CLUSTER_INFO:
                print title

    return DECIMALS.format(float(overall_accurate) / overall_total * 100)

#
   ====================================================================================

#
   ====================================================================================

#
   ====================================================================================

#
   ====================================================================================

#
   ====================================================================================

program_start = timer()
print "---------------- Dataset Loading"
print "Dataset:", CURRENT_DATASET, ", Feature:", FEATURE, ", Algorithm: ",
   ALGORITHM

if CURRENT_DATASET == "MNIST":
    if SHOW_CLUSTER_IMAGES:
        plt.gray() # MNIST images are grayscale

    start_time = timer()
    (training, training_lbls) = load_mnist(dataset="training", path="mnist"
       )
    (testing, testing_lbls) = load_mnist(dataset="testing", path="mnist")
    dataset_label_names = None
    dataset_filenames = None

    if DEBUGGING: # 60,000 max
        training = training[0:500]
        training_lbls = training_lbls[0:500]
```

```
        testing = testing[0:500]
        testing_lbls = testing_lbls[0:500]

elif CURRENT_DATASET == "CIFAR10":
    print "Loading", CIFAR_BATCHES, "CIFAR-10 batches"
    start_time = timer()
    # Load the first batch
    (dataset, dataset_lbls, dataset_label_names, dataset_filenames) =
       load_cifar(batch_name="data_batch_1", path="cifar")

    # Append all the batches requested into one dataset
    for index in range(2, CIFAR_BATCHES+1):
        (dataset2, dataset_lbls2, dataset_label_names2, dataset_filenames2)
            = load_cifar(batch_name="data_batch_"+str(index), path="cifar")

        dataset = np.append(dataset, dataset2, axis=0)
        dataset_lbls = np.append(dataset_lbls, dataset_lbls2, axis=0)
        dataset_label_names = np.append(dataset_label_names,
           dataset_label_names2, axis=0)
        dataset_filenames = np.append(dataset_filenames, dataset_filenames2
           , axis=0)

    if DEBUGGING:
        training = dataset[0:200]
        training_lbls = dataset_lbls[0:200]

        testing = dataset[200:400]
        testing_lbls = dataset_lbls[200:400]
    else:
        # Split into training and testing sets
        split = int(len(dataset)*0.75)
        training = dataset[:split]
        training_lbls = dataset_lbls[:split]

        testing = dataset[split:]
        testing_lbls = dataset_lbls[split:]

else:
    raise Exception("CURRENT_DATASET is invalid. Only [MNIST, CIFAR10] are
       valid datasets.", CURRENT_DATASET)

print "Took ", DECIMALS.format(timer() - start_time) , "s to load and split
   ", CURRENT_DATASET
print "Training:", training.shape[0], ",Testing:", testing.shape[0]

#
   ================================================================================

print "---------------- Computing Features:", FEATURE
start_time = timer()
```

```
if FEATURE == "PIXEL":
    training_flat = pixel_intensity(training, PERFORM_DESKEW)
    testing_flat = pixel_intensity(testing, PERFORM_DESKEW)

elif FEATURE == "COLORHIST":
    training_flat = color_histogram(training, HISTOGRAMS)
    testing_flat = color_histogram(testing, HISTOGRAMS)

elif FEATURE == "HOG":
    training_flat = gradient_histogram(training, HISTOGRAMS)
    testing_flat = gradient_histogram(testing, HISTOGRAMS)

elif FEATURE == "DAISY":
    training_flat = daisy_feature(training)
    testing_flat = daisy_feature(testing)

else:
    raise Exception("FEATURE is invalid. Only [PIXEL, COLORHIST, HOG, DAISY
       ] are valid features.", FEATURE)

if FEATURE_SCALING:
    scaler = StandardScaler()
    scaler.fit(training_flat)
    training_flat = scaler.transform(training_flat)

    scaler.fit(testing_flat)
    testing_flat = scaler.transform(testing_flat)

print "Took ", DECIMALS.format(timer() - start_time) , "s to compute
   features"
#
   ================================================================================

print "---------------- Training:", ALGORITHM
print "Performing",ALGORITHM,"training for {0} images.".format(training.
   shape[0])
start_time = timer()

if ALGORITHM == "AFFINITYPROP":
    k_means = AffinityPropagation(affinity="euclidean")
    k_means.fit(training_flat)
    CLUSTERS = len(k_means.cluster_centers_)

elif ALGORITHM == "MINIBATCHKMEANS":
    k_means = MiniBatchKMeans(n_clusters=CLUSTERS)
    k_means.fit(training_flat)

elif ALGORITHM == "MEANSHIFT":
    k_means= MeanShift()
    k_means.fit(training_flat)
    CLUSTERS = len(k_means.cluster_centers_)
```

```
elif ALGORITHM == "MLP":
    k_means = MLPClassifier(hidden_layer_sizes=(NEURONS, ),
                            activation='relu', solver='adam',
                            alpha=0.0001, batch_size='auto',
                            learning_rate='constant', learning_rate_init
                               =0.001,
                            power_t=0.5, max_iter=10, shuffle=True,
                            random_state=None, tol=0.0001, verbose=False,
                            warm_start=False, momentum=0.9,
                               nesterovs_momentum=True,
                            early_stopping=False, validation_fraction=0.1,
                            beta_1=0.9, beta_2=0.999, epsilon=1e-08)

    k_means.fit(training_flat, training_lbls.flatten())

    print "Neurons: ", NEURONS

elif ALGORITHM == "MLP2":
    k_means = MLPClassifier2(
        layers=[
            Layer("Rectifier", units=NEURONS),
            Layer("Softmax")],
        learning_rate = 0.1,
        learning_rule='nesterov',
        learning_momentum=0.9,
        batch_size=300,
        valid_size=0.0,
        f_stable=0.001,
        n_stable=10,
        n_iter=10)
    k_means.fit(training_flat, training_lbls.flatten())

    print "Neurons: ", NEURONS

elif ALGORITHM == "CNN":
    k_means = MLPClassifier2(
        layers=[
            #Native(lasagne_l.Conv2DLayer(incoming=(32,3,3), num_filters
               =32, filter_size=(3, 3)), nonlinearity=lasagne_nl.rectify),
            Convolution('Rectifier',channels=32,kernel_shape=(3,3),
               border_mode="same"),
            Native(lasagne_l.DropoutLayer,p=0.2),
            #Native(lasagne_l.Conv2DLayer,num_filters=32, filter_size=(3,
               3)),
            Convolution('Rectifier',channels=32,kernel_shape=(3,3),
               border_mode="same"),
            Native(lasagne_l.MaxPool2DLayer,pool_size=(2,2)),
            Native(lasagne_l.FlattenLayer),
            Native(lasagne_l.DenseLayer,num_units=NEURONS),
            Native(lasagne_l.DropoutLayer,p=0.5),
```

```
            #Convolution('Rectifier', channels=24, kernel_shape=(3, 3),
               dropout=0.05),
            #Layer('Rectifier', units=NEURONS, dropout=0.05),
            Layer('Softmax'),
            ],
        learning_rate=0.01,
        learning_rule='sgd',
        learning_momentum=0.9,
        batch_size=32,
        weight_decay=0.01/EPOCHS,
        valid_size=0.0,
        f_stable=0.001,
        n_stable=10,
        n_iter=EPOCHS)

    k_means.fit(training_flat, training_lbls.flatten())

    print "Neurons: ", NEURONS

elif ALGORITHM == "LINEARSVC":
    k_means = LinearSVC()
    k_means.fit(training_flat, training_lbls.flatten())

elif ALGORITHM == "SVC9":
    k_means = SVC(degree=9)
    k_means.fit(training_flat, training_lbls.flatten())

else:
    k_means = KMeans(n_clusters=CLUSTERS, n_jobs=1)
    k_means.fit(training_flat)


print ALGORITHM, "took ",DECIMALS.format(timer() - start_time), "s"

#
   ================================================================================

print "---------------- ",ALGORITHM,"Prediction"
print "Performing ",ALGORITHM," prediction for {0} images.".format(testing.
   shape[0])
start_time = timer()

predict_lbls = k_means.predict(testing_flat)

print "Took ",DECIMALS.format(timer() - start_time),"s"

#
   ================================================================================

print "----------------",ALGORITHM,"Testing Accuracy Evaluation"
```

```
start_time = timer()

if ALGORITHM == "MLP" or ALGORITHM == "LINEARSVC" or ALGORITHM == "CNN" or
   ALGORITHM == "SVC9" or ALGORITHM == "MLP2" or ALGORITHM == "AUTOENCODER"
   :
    testing_accuracy = k_means.score(testing_flat, testing_lbls) * 100
else:
    testing_accuracy = evaluate_clustering(testing, testing_flat,
       predict_lbls, testing_lbls, k_means.cluster_centers_,
       dataset_label_names)

print ALGORITHM, "Testing Accuracy:", testing_accuracy, "%"
print "Took ",DECIMALS.format(timer() - start_time),"s"

print "Program ran for ",DECIMALS.format(timer() - program_start), "s"
```

lib_features.py

```
# -*- coding: utf-8 -*-
"""
Course: CS 4365/5354 [Computer Vision]
Author: Jose Perez [ID: 80473954]
Assignment: Lab 4
Instructor: Olac Fuentes
"""
import numpy as np
from scipy import signal
from skimage.feature import daisy
from skimage import color
# conda install -c https://conda.binstar.org/menpo opencv
import cv2

#
   ================================================================================

# ============================== Pixel Intensity
def pixel_intensity(I, perform_deskew=False):
    if len(I.shape) == 3: # MNIST
        (total, w, h) = I.shape

        if perform_deskew:
            d1 = deskew(I[0])
            (w, h) = d1.shape
            temp = np.zeros((I.shape[0], w, h))
            for index in range(1, I.shape[0]):
                temp[index] = deskew(I[index])
            I = temp

        return np.reshape(I, (total, w * h))
    elif len(I.shape) == 4: # CIFAR
        (total, w, h, c) = I.shape
```

```
        return np.reshape(I, (total, w * h * c))

    raise Exception("[pixel_intensity] Invalid image dimensions: ", I.shape
       )

SZ = 20
affine_flags = cv2.WARP_INVERSE_MAP|cv2.INTER_LINEAR

def deskew(img):
    m = cv2.moments(img)
    if abs(m['mu02']) < 1e-2:
        return img.copy()
    skew = m['mu11']/m['mu02']
    M = np.float32([[1, skew, -0.5*SZ*skew], [0, 1, 0]])
    img = cv2.warpAffine(img,M,(SZ, SZ),flags=affine_flags)
    return img

#
   ================================================================================

# ============================== Histogram of Gradients
def gradient_histogram(I, number_of_bars):
    histograms = np.zeros((I.shape[0], number_of_bars * 4))

    for index in range(I.shape[0]):
        histograms[index] = cv_hog(I[index], number_of_bars)
            #if len(I.shape) == 4: # CIFAR
                #histograms[index] = regular_color_gradient_histogram(I[
                   index], number_of_bars)
            #elif len(I.shape) == 3: # MNIST
                #histograms[index] = regular_grayscale_gradient_histogram(I
                   [index], number_of_bars)
            #else:
                #raise Exception("[gradient_histogram] Invalid image
                   dimensions: ", I.shape)

    return histograms
# http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml
   /py_svm/py_svm_opencv/py_svm_opencv.html
# Histogram of Gradients using Sobel
# Divides region into 4 subregions
def cv_hog(img, number_of_bars):
    gx = cv2.Sobel(img, cv2.CV_64F, 1, 0)
    gy = cv2.Sobel(img, cv2.CV_64F, 0, 1)
    mag, ang = cv2.cartToPolar(gx, gy)

    # quantizing binvalues in (0...16)
    bins = np.int32(number_of_bars*ang/(2*np.pi))

    # Divide to 4 sub-squares
    bin_cells = bins[:10,:10], bins[10:,:10], bins[:10,10:], bins[10:,10:]
```

```
    mag_cells = mag[:10,:10], mag[10:,:10], mag[:10,10:], mag[10:,10:]

    hists = [np.bincount(b.ravel(), m.ravel(), number_of_bars) for b, m in
       zip(bin_cells, mag_cells)]
    hist = np.hstack(hists)
    return hist
#
   ================================================================================

# ============================== Color Histogram
def color_histogram(I, number_of_bars):
    if len(I.shape) == 4: # CIFAR Only
        (total, w, h, c) = I.shape
        histograms = np.zeros((total, 3, number_of_bars+1))
        for index in range(total):
            histograms[index] = regular_color_histogram(I[index],
               number_of_bars)

        return histograms.reshape(total, 3 * (number_of_bars+1))

    raise Exception("[color_histogram] Invalid image dimensions: ", I.shape
       )

def regular_color_histogram(I, number_of_bars):
    hist = np.zeros((3, number_of_bars+1))
    Iq = np.uint8(I * float(number_of_bars) / 256.0)

    for bucket in range(number_of_bars):
        hist[0, bucket] = I[Iq[:,:,0] == bucket].sum()
        hist[1, bucket] = I[Iq[:,:,1] == bucket].sum()
        hist[2, bucket] = I[Iq[:,:,2] == bucket].sum()

    return hist

#
   ================================================================================

# ============================== Daisy Features
def daisy_feature(I):
    if len(I.shape) == 4: # CIFAR
        (total, w, h, c) = I.shape
        RADIUS = 15

    elif len(I.shape) == 3: # MNIST
        (total, w, h) = I.shape
        RADIUS=1

    else:
        raise Exception("[daisy_feature] Invalid image dimensions: ", I.
           shape)
```

```
    # Calculate the first vector to find out the dimensions
    first_daisy = daisy(color.rgb2gray(I[0]), radius=RADIUS)
    (p, q, r) = first_daisy.shape

    # Create a place to store the features
    daisy_vector = np.zeros((total, p, q, r))
    # Add the already calculated feature
    daisy_vector[0] = first_daisy

    for index in range(1, total):
        daisy_vector[index] = daisy(color.rgb2gray(I[index]), radius=RADIUS
           )

    (total, p, q, r) = daisy_vector.shape
    return np.reshape(daisy_vector, (total, p * q * r))
```

lib_mnist.py

```
# -*- coding: utf-8 -*-
"""
Course: CS 4365/5354 [Computer Vision]
Author: Gustav Larsson (http://g.sweyla.com/blog/2012/mnist-numpy/)
Assignment: Lab 4
Instructor: Olac Fuentes
"""
import os, struct
import numpy as np
from array import array as pyarray
from numpy import array, int8, uint8, zeros

def load_mnist(dataset="training", digits=np.arange(10), path='.'):
    """
    Loads MNIST files into 3D numpy arrays

    Adapted from: http://abel.ee.ucla.edu/cvxopt/_downloads/mnist.py
    """

    if dataset == "training":
        fname_img = os.path.join(path, 'train-images.idx3-ubyte')
        fname_lbl = os.path.join(path, 'train-labels.idx1-ubyte')
    elif dataset == "testing":
        fname_img = os.path.join(path, 't10k-images.idx3-ubyte')
        fname_lbl = os.path.join(path, 't10k-labels.idx1-ubyte')
    else:
        raise ValueError("dataset must be 'testing' or 'training'")

    flbl = open(fname_lbl, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    lbl = pyarray("b", flbl.read())
    flbl.close()
```

```
    fimg = open(fname_img, 'rb')
    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    img = pyarray("B", fimg.read())
    fimg.close()

    ind = [ k for k in range(size) if lbl[k] in digits ]
    N = len(ind)

    images = zeros((N, rows, cols), dtype=uint8)
    labels = zeros((N, 1), dtype=int8)
    for i in range(len(ind)):
        images[i] = array(img[ ind[i]*rows*cols : (ind[i]+1)*rows*cols ]).
           reshape((rows, cols))
        labels[i] = lbl[ind[i]]

    return images, labels
```

lib_cifar.py

```
# -*- coding: utf-8 -*-
"""
Course: CS 4365/5354 [Computer Vision]
Author: Chelsey J (with some minor modifications by Jose Perez)
Assignment: Lab 4
Instructor: Olac Fuentes
"""
import os
import cPickle
import numpy as np

def unpickle(file):
    fo = open(file, 'rb')
    dict = cPickle.load(fo)
    fo.close()
    return dict

def load_cifar(batch_name="data_batch_1", path="."):
    """
    Loads CIFAR-10 files to 3d numpy arrays
    """
    file = os.path.join(path, batch_name)
    batches = os.path.join(path, 'batches.meta')

    data = unpickle(file).get('data')
    filename = np.array(unpickle(file).get('filenames'))
    labels = np.array(unpickle(file).get('labels'))
    label_names = np.array(unpickle(batches).get('label_names'))

    im_num = data.shape[0]
    row, col = 32, 32
    images = np.zeros((im_num, row, col, 3))
```

```
    for im in range(im_num):
        r_channel = data[im][0:1024].reshape((32, 32))
        g_channel = data[im][1024:2048].reshape((32, 32))
        b_channel = data[im][2048:3072].reshape((32, 32))
        images[im] = np.dstack([r_channel, g_channel, b_channel])

    return images, labels, label_names, filename
```

B. Academic Honesty Certification

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

Jose Perez

Date