Using SIFT Descriptors for Image Classification

Nuno Granja Fernandes
up201107699

Samuel Arleo Rodriguez
up201600802

Katja Hader
up201602072

Universidade do Porto
Faculdade de Engenharia
FEUP

Abstract

The goal of this project is to develop a system that is able to analyse and classify an image correctly. The project is based on an existing kaggle competition so we used the given training and testing set of images as well as the *csv* files containing the labels.

1 Introduction

The main steps used to build a system capable of classifying images were:

- Extraction of *keypoints* and *descriptors* of the images in the training set.
- Creating a visual vocabulary from the descriptors of all the images
- Assembling a “bag of words” to each image
- Applying a multi-class classifier, using the bag of words of the image to classify and comparing it to the existing bags of words from the training set.

Additionally, there was the necessity to create classes for “descriptor”, “image” and “images set”, being that and “image” will contain one or more “descriptor” objects and “images set” will contain many “image” objects.

1.1 Feature extraction

In computer vision, local descriptors (features computed over limited spatial support) have proved well-adapted to matching and recognition tasks, as they are robust to partial visibility and clutter. Such tasks require descriptors that are repeatable. Here, we mean repeatable in the sense that if there is a transformation between two instances of an object, corresponding points are detected and identical descriptor values are obtained around each one of them.

Specifically, our system uses SIFT to detect *keypoints* and extract *descriptors* from both the training and testing image data sets. Followed by assigning each image the respective tuple (keypoints, descriptors) as attribute.

1.2Image representation

Our bag of keypoints approach can be motivated by an analogy to learning methods using the bag-of-words representation for text categorization.

This way, a “KMeans” algorithm is applied to all the descriptors from the training images set, in order to obtain 150 clusters, each cluster representing a word. This way, the “visual vocabulary” is built with 150 different words. Next, the respective label, resulting of the “KMeans” algorithm, is assigned to its descriptor.

A method “classify_desc” of the object representing the set of test images is called to classify each image descriptors as “belonging” to one of the 150 available visual words. This is achieved through the use of the KNN (K Nearest Neighbours) algorithm, with K = 1, so that each descriptor is coupled it’s nearest neighbour.

When this process is over, a “bag of words” is built for each image in the test data set.

1.3 Classification

Once descriptors have been assigned to clusters to form feature vectors, we reduce the problem of generic visual categorization to that of multi-class supervised learning, with as many classes as defined visual categories. The classifier performs two separate steps in order to predict the classes of “unlabelled” images: training and testing.

To be able to classify a test image, first, a matrix of the bag of words of all the training images is assembled, followed by the matrix of labels, in which each line of the label matrix, describes the bag of words in the same line of the bag of words matrix. These two matrices are required in order to correctly train the chosen classifier.

The KNN classifier is then trained and is now able to classify all the test image’s bag of words.

1.4 Evaluation

No comparison was done to other methodology in the scope of this project, even though it would be expected to notice differences in execution time between feature extractors like SIFT and other faster algorithms like FAST or SURF. It also would be interesting to compare other classifiers such as Support Vector Machines (SVM).

1.5 Conclusion

This project proved very helpful in showing the capabilities of a system able to detect and classify different objects, either in a static image or video. We also became more aware of all the different feature detectors and extractors and in which circumstances each one of them should be used, such as the methodology used in different classifiers like SVM, KNN and Haar Cascades.

References

Bradski, G. and Kaehler, A. (2008). *Learning OpenCV*. 1st ed. Sebastopol, CA: O'Reilly.

Docs.opencv.org. (2016). *K-Nearest Neighbors — OpenCV 3.0.0-dev documentation*. [online] Available at: http://docs.opencv.org/3.0-beta/modules/ml/doc/k_nearest_neighbors.html [Accessed 31 Dec. 2016].

Annex

1. Source Code

```
##############################################################################################################
#   University of Porto
#   Faculty of Engineering
#   Computer Vision
#
# Project 2: Objects classification
#
# Authors:
#   * Katja Hader up201602072
#   * Nuno Granja Fernandes up201107699
#   * Samuel Arleo Rodriguez up201600802
##############################################################################################################

import numpy as np
import cv2
from os import listdir
from os.path import isfile, join, splitext
import csv
import classification, representation, featurex
import time
from collections import Counter

class descriptor:
    def __init__(self, vector=None, label=None):
        self.vector = vector
        self.label = label

class image:
    def __init__(self, img=None, name=None, keyp=None, desc=None, hist=None):
        self.img = img
        self.id = None
        self.label = None           # Image class
        self.desc = desc            # Array of descriptor objects
        self.keyp = keyp            # Array of keypoints
        self.histogram = hist
        self.name = name

    # Given a list of labels, assign them to their correspondent descriptor and computes img histogram
    def set_labels(self,labels,desc_size):
        self.histogram = np.zeros(desc_size, dtype=object)
        for i in range(0,len(labels)):
            self.desc[i].label = labels[i]
            self.histogram[label[i]] += 1
class images_set:
    def __init__(self, path_img=None, path_lab=None):
        self.images = None
        self.path_img = path_img
        self.path_lab = path_lab

    def load_images(self,inf,sup):
        try:
            # Creating a list with all pictures names
            onlyfiles = [ f for f in listdir(self.path_img) if isfile(join(self.path_img,f)) ]
            onlyfiles = sorted(onlyfiles)[inf:sup]
            self.images = np.empty(len(onlyfiles), dtype=object)
            # Reading each image and storing it into the images array
            for n in range(0, len(onlyfiles)):
                name = onlyfiles[n]
                self.images[n] = image(cv2.imread(join(self.path_img,name),0),int(splitext(name)[0]))
        except:
            print "Error opening the folder ",self.path_img,".Please check the file location."
            exit()

    def load_labels(self,inf,sup):
        try:
            # Loading labels into an matrix with columns |id|label|
            labels = np.genfromtxt(self.path_lab, delimiter=',',
                dtype=[('id','<i8'),('label','|S5')], skip_header=1)
        except:
            print "Error opening the file ",self.path_lab,".Please check the file location."
            exit()
        n = 0
        # Adding id and label to each image
        for (x,y) in labels[inf:sup]:
            self.images[n].id = x
            self.images[n].label = y
            if n == len(self.images)-1:
                break
            n += 1

    # Gets the list of descriptors (not descriptor objects yet) and returns a list of desc. objects
    # that have the actual descriptor in the vector attribute
    def build_desc(self, desc_list):
        return map(lambda x: descriptor(x), desc_list)                  # For each desc in desc_list map replaces it
                                                                        # by a descriptor object
    # Assings descriptors and keypoints to their correspondent image
    # Args: features contains pairs of |Keypoints| Descriptors|
    def get_features(self, features):
        for i in range(0,len(features)):
            if features[i][1] is not None:                              # To discard pictures without keypoints
                self.images[i].desc = self.build_desc(features[i][1])   # Assign to each image a list of desc. objects
                self.images[i].keyp = features[i][0]
            #else:
            #   self.images[i].desc = np.array([])

    # Computes the histogram of each image and assigns the label to each descriptor
    def set_descr_labels(self,labels,desc_size,bag_size):
        index = 0
        for img in self.images:                                         # For each image in the set
            img.histogram = np.zeros(desc_size, dtype=object)           # Initializes the histogram vector (250 words so far)
            if img.desc is not None:                                    # Discard images without keypoints
                for desc in img.desc:                                   # For each descriptor on each image
                    label = labels[index][0]                            # Stores the label of the current descriptor
                    desc.label = label                                  # Assigns to the descriptor its label
                    img.histogram[label] += 1                           # Adds 1 to the element in the position of the
                    index += 1                                          # label value. Ej label = 3, histogram[3] += 1
            else:
                img.histogram = np.zeros(bag_size,dtype=np.float32)

    # Classify descriptors of test images
    def classify_desc(self, centers, bag_size):
        desc_size = centers.shape[0]
        labels = np.linspace(0,desc_size-1,num=desc_size,
                                dtype=np.int32).reshape(-1,1)  # Array with labels from (0 to 249)
        knn = cv2.ml.KNearest_create()
        knn.train(centers,cv2.ml.ROW_SAMPLE,labels)
        for img in self.images:
            if img.desc is not None:
                desc = np.array(map(lambda x: x.vector, img.desc))      # Putting together descriptors of each image (x = descriptor)
                ret,result,neighbours,dist = knn.findNearest(desc,k=1)
                img.set_labels(desc, desc_size)                         # Set labels of descriptors of img
            else:
                img.histogram = np.zeros(bag_size,dtype=np.float32)

# Stores all descriptors of the set of images in a single variable to cluster them
def join_desc(res):
    # res has columns |Keypoint|Descriptors| and each row represent a keypoint
    # tmp stores just the descriptors
    tmp = [res[i][1] for i in range(0,len(res)) if res[i][1] is not None]
    # Getting descriptors size (all have the same given by SIFT: 128)
    desc_size = tmp[0][0].shape[0]
    # Counting number of descriptors
    num_desc = 0
    for img in tmp:
        for desc in img:
            num_desc += 1
    # Storing descriptors in des, but before we initialze it empty with the right dimensions: [num_desc,128]
    des = np.zeros((num_desc,desc_size))
    n = 0
    for img in tmp:
        for desc in img:
            des[n,:] = desc
            n += 1
    return des

#----------------------- LOADING DATA --------------------------

# Paths to the training and test data
```

```
path_train_imgs = "/home/samuel/CV2/train_data/"
path_test_imgs = "/home/samuel/CV2/train_data/"

# File with labels of training images
train_labels = "/home/samuel/CV2/labels_train.csv"
test_labels = "/home/samuel/CV2/labels_train.csv"

# Using a subset of the images set
inf_tr = 0
sup_tr = 9000
inf_ts = 9000
sup_ts = 10000

# Creating object images_set that encapsulates methods for loading images and labels,
# and also stores the loaded images and labels
train_set = images_set(path_train_imgs, train_labels)
test_set = images_set(path_test_imgs, test_labels)

# Loading sup-inf number of images
train_set.load_images(inf_tr, sup_tr)
test_set.load_images(inf_ts, sup_ts)

# Loading labels of previously loaded pictures
train_set.load_labels(inf_tr,sup_tr)
test_set.load_labels(inf_ts,sup_ts)

#------------------- EXTRACTING FEATURES -----------------------

# Instantiating sift class
sift = cv2.xfeatures2d.SIFT_create()

# Applying SIFT to all training images. This returns the tuple (keypoints, descriptor)
# for each image, and it's transformed to a matrix with columns:
# |Keypoints| Descriptors|
res_train = map(lambda x: sift.detectAndCompute(x.img, None), train_set.images)
res_test = map(lambda x: sift.detectAndCompute(x.img, None), test_set.images)

# Storing each descriptor and keypoint with its image
train_set.get_features(res_train)
test_set.get_features(res_test)

# Storing all descriptors of training images in a single variable to cluster them
desc = join_desc(res_train)

# Changing type to float32 which is required by the kmeans function
desc = desc.astype('float32')

# Parameters of the k-means algorithm
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

# Number of cluster that will set the number of words of each bag
words_number = 150

# Measuring time of k-means
start = time.time()

# Applying k-means to all the descriptors
ret,label,centers=cv2.kmeans(desc,words_number,None,criteria,4,cv2.KMEANS_RANDOM_CENTERS)

# Computing and showing k-means run time
end = time.time()
print(end - start)

#------------------- REPRESENTATION STEP -----------------------

# Giving a label to each descriptor. Also passing size of descriptors: desc[0].shape[0]
train_set.set_descr_labels(label, words_number, words_number)

# Classifying descriptors of test images. Each descriptor can belong to 250 classes, i.e,
# each new word will be more similar to one of the 250 visual words computed in k-means
test_set.classify_desc(centers, words_number)

#------------------- CLASSIFYING STEP ------------------------

# Putting together all the bag of words
bw_train = np.array(map(lambda x: x.histogram, train_set.images),dtype=np.float32)
bw_test = np.array(map(lambda x: x.histogram, test_set.images),dtype=np.float32)
```

```
# Creating arrays with labels to pass them to the predictors
labels_tr = np.array(map(lambda x: x.label,train_set.images))
labels_ts = np.array(map(lambda x: x.label,test_set.images))

# Images IDs
ids_tr = np.array(map(lambda x: x.label,train_set.images))

# Changing format of labels to int
classes, numeric_tr = np.unique(labels_tr, return_inverse=True)
numeric_tr = (numeric_tr).astype(np.int32)

# Training kNN with bag of words of the training set
knn = cv2.ml.KNearest_create()
knn.train(bw_train,cv2.ml.ROW_SAMPLE,numeric_tr)

# Predicting image class
ret,result,neighbours,dist = knn.findNearest(bw_test,k=1)

result = result.astype(np.int32)

pos = 0
neg = 0
for i in (classes[result]==labels_ts.reshape(-1,1)):
    if i:
        pos += 1
    else:
        neg += 1

print("POS: ",pos," NEG: ",neg)
print(float(pos)/float(neg))
#print(classes[result.astype(np.int32)] == labels_ts)
```