Object Recognition and Image Understanding
Exercise Sheet 12

Sascha Stelling

July 13, 2018

1

Code (see ism.py):

```
#!/usr/bin/env python2

import numpy as np
from skimage.feature import hog
from skimage.transform import resize

from helper_functions import *

if __name__ == "__main__":
path = "data/ETHZ/Applelogos/train/"
images, numImages = readImages(path, ".jpg")
bboxes = readBboxes(path)
crops = fillCrops(images, bboxes)
patcheslist = []

for crop in crops:
showImage(crop)
for crop in crops:
patcheslist.append(cropPatches(crop, step_size=8))

print "test"
codebook = []
for patches in patcheslist:
for patch in patches:
try:
patch_rescaled = resize(patch, (40,40), anti_aliasing=False)
codebook.append(hog(patch_rescaled))
except ValueError:
print patch.shape
```

Code (helper_functions.py):

```
#!/usr/bin/env python2

from __future__ import with_statement
from skimage.io import imread
import os
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import resize
import sys



def readImages(path, filetype):
# Read Images from path with each (sub)directory
# returns: list of images found, number of images
images = []
print "\nReading images from " , path

# Go through all subdirs to find image files
for dirpath, dirs, files in os.walk(path, topdown=True):
for filename in files:
if filename.endswith(filetype):
imagePath = os.path.join(dirpath, filename)
image = imread(imagePath, as_gray=True)
if image is None:
print "Image " , format(imagePath) , " not read properly!"
else:
# Convert image to floating point
image = np.float32(image)/255.0
images.append(image)

numImages = len(images)

if numImages == 0:
print "No Images found! Aborting..."
sys.exit(0)

print "Done! " , numImages , " images found."

return images, numImages

def readBboxes(path):
# Read bounding boxes from groundtruth files
# returns: list of bounding boxes
```

```
crops = []
print "\nReading bounding box data from " , path
for dirpath , dirs , files in os.walk(path , topdown=True):
for filename in files:
if filename.endswith(".groundtruth"):
cropPath = os.path.join(dirpath , filename)
with open(cropPath , 'r') as file:
bbox = file.read()
if bbox is None:
print "Groundtruth box coordinates not found at path " , format(imagePat
else:
nrs = []
for z in bbox.split(" "):
try:
nbr = float(z)
isnbr = True
except ValueError:
isnbr = False
pass
if isnbr:
# Cast to int since pixel coordinates can't be float
nrs.append(int(nbr))

crops.append(nrs)

if len(crops) == 0:
print "No groundtruth box coordinates found! Aborting..."
sys.exit(0)

print "Done! ", len(crops), " bounding box data files found."

return crops

def cropImage(img, bbox):
# Crops image according to bounding boxes
# Bounding box coordinates must be in order [x_min, y_min, x_max, y_max]
# and mustn't be greater than image size
# returns: cropped image

crop_img = img[bbox[1]:bbox[3], bbox[0]:bbox[2]]

return crop_img

def fillCrops(images, bboxes):
# Crop all images according to bounding boxes.
# Also consider multiple bounding boxes per image
# returns: list of cropped images

crops = []
print ”\nCropping images...”

for i in range(len(images)):
if len(bboxes[i]) == 4:
crop_img = cropImage(images[i], bboxes[i])
#print i, crop_img.shape, bboxes[i], images[i].shape
crops.append(crop_img)
elif len(bboxes[i]) > 4 and len(bboxes[i]) % 4 == 0:
newbbox = [bboxes[i][x:x+4] for x in range(0, len(bboxes[i]), 4)]
for j in range(0, len(newbbox)):
crop_img = cropImage(images[i], newbbox[j])
#print i, crop_img.shape, newbbox[j], images[i].shape
crops.append(crop_img)
else:
print ”Error: unmatched bounding box: ”, bboxes[i]

print crops[34]
print ”Finished cropping images. ”, len(crops), ” cropped Images created
return crops

def cropPatches(img, step_size):
# Crop an image into 8x8 grid
# Patches are padded to square shape
# returns: list of cropped patches (64 elements)
maxshape = max(img.shape)
diff_x = maxshape-img.shape[1]
diff_y = maxshape-img.shape[0]

if maxshape == img.shape[0]:
if diff_x % 2 == 0:
img = np.pad(img, ((0,0), (diff_x/2, diff_x/2)), mode='edge')
else:
img = np.pad(img, ((0,0), ((diff_x/2+1), diff_x/2)), mode='edge')

else:
if diff_y % 2 == 0:
img = np.pad(img, (((diff_y/2), diff_y/2),(0,0)), mode='edge')
else:
img = np.pad(img, (((diff_y/2)+1, diff_y/2),(0,0)), mode='edge')

assert img.shape[0] == img.shape[1]

sample_points = []
```

```
#print "\nCropping patches with step size ", step_size
for c1 in range(step_size, maxshape, maxshape//step_size):
for c2 in range(step_size, maxshape, maxshape//step_size):
sample_points.append([c1, c2])

patches = []
for point in sample_points:
patches.append(cropImage(img, [point[0]-step_size, point[1]-step_size, p

#print "Finished cropping patches."
return patches

def showImage(img):
plt.imshow(img, cmap="gray")
plt.show()
```

2

1. Code (see optical_flow.py):

```
#!/usr/bin/env python2

from helper_functions import readImages, showImage
from functions_lib import compute_optical_flow
import numpy as np

if __name__ == "__main__":
images, numImages = readImages("data/Crowd_PETS09", ".jpg")
flow = mag = ang = [0] * len(images)
for i in range(1, len(images)):
flow[i], mag[i], ang[i] = compute_optical_flow(images[i], im
```

2. Drawbacks: The algorithm assumes that the pixel intensity does not change between consecutive frames which means moving objects which experience a change in illumination will not be detected anymore and the detection depends a lot on the lighting. It also assumes that neighbouring pixels have similar motion which results in detection problems when two objects are next to each other and begin to overlap on the image.
   To address the first issue, you could use an intensity invariant detection model like SIFT to detect objects in the scene.
   For the second issue, you could use segmentation to get the outline for every object and try to maintain this outline even when another object crosses.