Analyse et Traitement d'images

ENSSAT

LANNION

PADE Cyril IMR3

23/02/2020

Sommaire

Introduction

Lors de ce module, différents méthodes ont été expliqués et utilisés sur des images. Elles ont pour but de classifier ces dernières selon des classes. Deux grandes familles de classification existent, nommées Old School et New School. Leurs fonctionnements sera expliqué par la suite, il est à savoir qu'il est également possible de laisser à la machine quelles classes sont à créer. Cette alternative ne sera pas étudiée ici, les classes seront toujours données avant traitement.

Le cadre

Le contexte dans lequel se sont produit les résultats sera expliqué, par le biais des outils et ressources utilisés dans les sous-parties à venir.

Le langage

Afin d'éviter de prendre du temps à développer certaines fonctionnalités, python a été choisi de son ergonomie et des packages qui disposent de celles-ci. Afin d'isoler l'environnement et éviter des problèmes de compatibilités, un venv a été créé sous python 3.6. Un venv permet de créer un contexte différent de celui par défaut pour l'utilisateur via les commandes suivantes :

```
# pour installer le package avec pip
pip install virtualenv
 # pour créer un venv
python3 -m venv env
# pour accéder à un venv
source env/bin/activate
# pour quitter un venv
deactivate
```

Les différents paquets présents dans le venv du projet sont disponibles dans le git(fichiers freeze), ainsi qu'en annexe.

La base de données

On parle de traitement et analyse d'image, mais pour cela il nous faut des images. Et un grand nombre afin de rendre cohérent l'automatisation de leur traitement, pour cela une

base destinées à cette usage a été utilisé. CorelDB est fournie avec 10800 images en 80 catégaories, disponible à l'url suivant: https://sites.google.com/site/dctresearch/Home/content-based-image-retrieval.

Le dispatcher

Afin de pouvoir travailler sur différents catégories d'images en trois étapes sont avoir de doublons, un dispatcher a été utilisé son code est le suivant :

```
from os import listdir, mkdir
from os.path import exists, isdir, isfile, join, splitext
from shutil import copyfile

global root, feature_number
global path_training, path_test, path_validation

root = './'
feature_number = 5
def to_int(s):
    try:
        return int(s)
    except ValueError:
        return None
def create_folders():
    global path_training, path_test, path_validation
    path_training = 'db_train'
    path_test = 'db_test'
    path_validation = 'db_valid'
    if not exists(path_training): mkdir(path_training)
    if not exists(path_test): mkdir(path_test)
    if not exists(path_validation): mkdir(path_validation)
    #print("create folders")
def create_features(path):
    feature_training = join(path_training, path)
    feature_test = join(path_test, path)
    feature_validation = join(path_validation, path)
    if not exists(feature_training): mkdir(feature_training)
    if not exists(feature_test): mkdir(feature_test)
    if not exists(feature_validation): mkdir(feature_validation)
    #print("create feature %s" % path)
```

```
def main():
    create_folders()
    feature_count = 1
    for feature in listdir(root):
        root_feature = join(root, feature)
        if feature_count <= feature_number and isdir(root_feature):
            feature_count += 1
            create_features(feature)
            for picture in listdir(root_feature):
                root_picture = join(root_feature, picture)
                if isfile(root_picture):
                    filename = splitext(picture)[0]
                    file_last_int = to_int(filename[len(filename)-1])
                    if file_last_int != None:
                        rel_path = join(feature, picture)
                        src = join(root, rel_path)
                        dst = ""
                        if file_last_int in [0,1,2]:
                            dst = join(path_training, rel_path)
                        elif file_last_int in [3,4,5,6]:
                            dst = join(path_test, rel_path)
                        else:
                            dst = join(path_validation, rel_path)
                        copyfile(src, dst)
if __name__ == "__main__":
```

Grâce à cela les bases d'entraînement (db_train), de validation (db_valid), de test (db_test) sont créées. Cela se fait en sélectionnant X classes ou catégories, ici 5 (feature_number) vont être choisies et réparties selon leur dernier digits. Le ratio pour db_train, db_valid, db_test est respectivement 30%,30%,40%.

Old School

Ici, on détermine sur quelle partie de l'image l'on va travailler comme par exemple la couleur, les bords, ...etc. Une base de code est disponible sur github, son auteur est Po-Chih Huang (pochih). Le code implémente les objets suivants :

- color.py
- daisy.py

- DB.py
- edge.py
- evaluate.py
- fusion.py
- gabor.py
- HOG.py
- infer.py
- random_projection.py
- resnet.py
- temp.py
- vggnet.py

Le dépot github est disponible à l'adresse suivante :

https://github.com/pochih/CBIR/tree/master/src

Seulement Color.py, Edge.py, Fusion.py, DB.py et evaluate.py cont être utilisés et/ou modifiés.

Evaluate

Une tentative de prédiction a été commenté pour non fonctionnement, et donc les modifications sont les suivantes :

```
diff --git a/src/evaluate.py b/src/evaluate.py
index a5bd656..1fa5468 100644
--- a/src/evaluate.py
+++ b/src/evaluate.py
@@ -54,14 +54,14 @@ def AP(label, results, sort=True):
   if sort:
     results = sorted(results, key=lambda x: x['dis'])
   precision = []
-  hit = 0
+  hit =1
   for i, result in enumerate(results):
     if result['cls'] == label:
       hit += 1
-      precision.append(hit / (i+1.))
+      precision.append(hit)
```

```
   if hit == 0:
     return 0.
-  return np.mean(precision)
+  return np.mean(precision)>0.50


 def infer(query, samples=None, db=None, sample_db_fn=None,
depth=None, d_type='d1'):
@@ -95,6 +95,7 @@ def infer(query, samples=None, db=None,
sample_db_fn=None, depth=None, d_type='d
     if q_img == s_img:
       continue
     results.append({
+                    'img': s_img,
                     'dis': distance(q_hist, s_hist, d_type=d_type),
                     'cls': s_cls
                   })
@@ -139,14 +140,40 @@ def evaluate_class(db, f_class=None,
f_instance=None, depth=None, d_type='d1'):

   classes = db.get_class()
   ret = {c: [] for c in classes}
-
+  import pandas as pd
+  retN=pd.DataFrame({'query':[],'N1':[],'N2':[],'N3':[]})
   if f_class:
     f = f_class()
   elif f_instance:
     f = f_instance
   samples = f.make_samples(db)
+  #return samples,samples
+  i=0
   for query in samples:
-    ap, _ = infer(query, samples=samples, depth=depth,
d_type=d_type)
+    ap, var = infer(query, samples=samples, depth=depth,
d_type=d_type)
     ret[query['cls']].append(ap)
-
```

```
-  return ret
+    print("iteration", i)
+    VARIABLE=[sub['img'] for sub in var]
+    # print ("i :"+str(i)+"/"+str(len(retN)))
+    print ("Variable :"+str(len(VARIABLE)))
+    if(len(VARIABLE)!=3):
+      continue
+    retN.loc[i]=[query['img'],VARIABLE[0],VARIABLE[1],VARIABLE[2]]
+    i=i+1
+  return ret, retN
+#  list_im = [ sub['img'] for sub in results]
+#  pred = [sub['cls'] for sub in results ]
+#  weight = [sub['dis'] for sub in results ]
+#  weight = np.reciprocal(weight)
+#  pred2 = weighted_mode(pred, weight)
+#  pred = np.array_str(pred2[0])[2:-2]
+#  list_im.insert(0, q_img)
+#  imgs = [Image.open(i) for i in list_im]
+#  min_shape = sorted([(np.sum(i.size), i.size) for i in
imgs])[0][1]
+#  imgs_comb = np.hstack((np.asarray(i.resize(min_shape)) for i in
imgs))
+#  plt.imshow(imgs_comb/255.)
+#  plt.pause(2)
+#  plt.close()
+#  ap = AP(q_cls, results, sort=False)
+#  return ap, pred
+#
```

DB

Les chemins ont été modifiés afin que les traitements utilisant ce bloc puissent récupérer les bonnes images.

```
diff --git a/src/DB.py b/src/DB.py
index 388a762..97cfd11 100644
--- a/src/DB.py
+++ b/src/DB.py
@@ -5,8 +5,8 @@ from __future__ import print_function
```

```
 import pandas as pd
 import os

-DB_dir = 'database'
-DB_csv = 'data.csv'
+DB_dir = '../../CorelDB/db_train'
+DB_csv = './data.csv'


 class Database(object):
@@ -18,7 +18,7 @@ class Database(object):

   def _gen_csv(self):
     if os.path.exists(DB_csv):
-      return
+      os.remove(DB_csv)
     with open(DB_csv, 'w', encoding='UTF-8') as f:
       f.write("img,cls")
       for root, _, files in os.walk(DB_dir, topdown=False):
@@ -43,6 +43,6 @@ if __name__ == "__main__":
   db = Database()
   data = db.get_data()
   classes = db.get_class()
-
   print("DB length:", len(db))
   print(classes)
+#  test =set(db.get_data.iterable)
```

Color

Ce traitement permet de trier les images selon leurs couleurs pour les classifier.

Les modification auront été :

```
diff --git a/src/color.py b/src/color.py
index b46c305..de9f66e 100644
--- a/src/color.py
+++ b/src/color.py
@@ -7,7 +7,8 @@ from DB import Database
```

```
 from six.moves import cPickle
 import numpy as np
-import scipy.misc
+#import scipy.misc
+import imageio
 import itertools
 import os

@@ -86,7 +87,8 @@ class Color(object):
     if isinstance(input, np.ndarray):  # examinate input type
       img = input.copy()
     else:
-      img = scipy.misc.imread(input, mode='RGB')
+      img = imageio.imread(input, pilmode='RGB')
+   #   img = scipy.misc.imread(input, mode='RGB')
     height, width, channel = img.shape
     bins = np.linspace(0, 256, n_bin+1, endpoint=True)  # slice bins
equally for each channel

@@ -161,8 +163,8 @@ if __name__ == "__main__":
   color = Color()

   # test normalize
-  hist = color.histogram(data.ix[0,0], type='global')
-  assert hist.sum() - 1 < 1e-9, "normalize false"
+  # hist = color.histogram(data.ix[0,0], type='global')
+  # assert hist.sum() - 1 < 1e-9, "normalize false"

   # test histogram bins
   def sigmoid(z):
@@ -188,7 +190,7 @@ if __name__ == "__main__":
   assert distance(hist, hist2, d_type='d2-norm') == 2, "d2 implement
failed"

   # evaluate database
-  APs = evaluate_class(db, f_class=Color, d_type=d_type,
depth=depth)
+  APs,NMLIST = evaluate_class(db, f_class=Color, d_type=d_type,
depth=depth)
```

```
   cls_MAPs = []
   for cls, cls_APs in APs.items():
     MAP = np.mean(cls_APs)
```

Les résultats obtenus

```
Class sc_indoor, MAP 0.85
Class bld_modern, MAP 0.9206349206349206
Class MAGEL, MAP 0.9444444444444444
Class wl_lion, MAP 0.9666666666666667
Class sc_, MAP 0.9333333333333333
MMAP 0.9230158730158731
```

Edge

Ici on se base sur les contours des formes afin de classifier, les traces des modifications sont les suivantes :

```
diff --git a/src/edge.py b/src/edge.py
index ecd618b..ef24b28 100644
--- a/src/edge.py
+++ b/src/edge.py
@@ -11,6 +11,7 @@ import scipy.misc
 from math import sqrt
 import os

+import imageio

 stride = (1, 1)
 n_slice  = 10
@@ -104,7 +105,7 @@ class Edge(object):
     if isinstance(input, np.ndarray):  # examinate input type
       img = input.copy()
     else:
-      img = scipy.misc.imread(input, mode='RGB')
+      img = imageio.imread(input, pilmode='RGB')
     height, width, channel = img.shape

     if type == 'global':
@@ -193,7 +194,7 @@ if __name__ == "__main__":
```

```
   assert edge_kernels.shape == (5, 2, 2)

   # evaluate database
-  APs = evaluate_class(db, f_class=Edge, d_type=d_type, depth=depth)
+  APs,NMList = evaluate_class(db, f_class=Edge, d_type=d_type,
depth=depth)
   cls_MAPs = []
   for cls, cls_APs in APs.items():
     MAP = np.mean(cls_APs)
```

Les résultats obtenues

```
Class bld_modern, MAP 0.9523809523809523
Class sc_, MAP 0.3333333333333333
Class MAGEL, MAP 0.28888888888888886
Class wl_lion, MAP 0.03333333333333333
Class sc_indoor, MAP 0.18333333333333332
MMAP 0.3582539682539682
```

Fusion

Ce traitement permet de combiner deux attributs de l'image pour la classification, on utilisera edge (contour des formes) et color (couleur via histogramme). Le code changé:

```
diff --git a/src/fusion.py b/src/fusion.py
index 08b440b..8d263f2 100644
--- a/src/fusion.py
+++ b/src/fusion.py
@@ -17,11 +17,13 @@ import numpy as np
 import itertools
 import os

+import imageio

 d_type   = 'd1'
 depth    = 30

-feat_pools = ['color', 'daisy', 'edge', 'gabor', 'hog', 'vgg',
'res']
+# feat_pools = ['color', 'daisy', 'edge', 'gabor', 'hog', 'vgg',
```

```
'res']
+feat_pools = ['color', 'edge']

 # result dir
 result_dir = 'result'
@@ -103,7 +105,7 @@ def evaluate_feats(db, N, feat_pools=feat_pools,
d_type='d1', depths=[None, 300,
   for combination in combinations:
     fusion = FeatureFusion(features=list(combination))
     for d in depths:
-      APs = evaluate_class(db, f_instance=fusion, d_type=d_type,
depth=d)
+      APs,NMList = evaluate_class(db, f_instance=fusion,
d_type=d_type, depth=d)
       cls_MAPs = []
       for cls, cls_APs in APs.items():
         MAP = np.mean(cls_APs)
@@ -137,8 +139,8 @@ if __name__ == "__main__":
   evaluate_feats(db, N=7, d_type='d1')

   # evaluate database
-  fusion = FeatureFusion(features=['color', 'daisy'])
-  APs = evaluate_class(db, f_instance=fusion, d_type=d_type,
depth=depth)
+  fusion = FeatureFusion(features=['color', 'edge'])
+  APs,NMList = evaluate_class(db, f_instance=fusion, d_type=d_type,
depth=depth)
   cls_MAPs = []
   for cls, cls_APs in APs.items():
     MAP = np.mean(cls_APs)
```

Les résultats obtenues

```
Class wl_lion, MAP 0.0
Class sc_, MAP 0.8666666666666667
Class sc_indoor, MAP 0.15
Class MAGEL, MAP 0.1
Class bld_modern, MAP 1.0
MMAP 0.42333333333333334
```

New School

Les méthodes de réseau de neurones vont être utilisé afin de générer un modèle suite à l'entraînement sur les données, celui-ci sera réutilisé dans leur exécution et complété au fil d'itérations (Epoch). Keras est utilisé afin de simuler le système de neurone.

Entrainement

Dans cette partie le réseau de neurones n'a pas encore de modèle, une base sur laquelle s'appuyer lors de l'analyse de nouvelles images. Pour ce faire on fait deux classifications la première sur la base d'entraînement, la dernière sur la base validation. A l'issue de cette procédure on génère un fichier qui sera réutilisé par la suite.

Code

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K

import os

os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

# dimensions of our images.
img_width, img_height = 64, 64

train_data_dir = '../CorelDB/db_train'
validation_data_dir = '../CorelDB/db_valid'

nb_classes = sum([len(d) for r, d, files in os.walk(train_data_dir)])
nb_train_samples = sum([len(files) for r, d, files in
os.walk(train_data_dir)])
nb_validation_samples = sum([len(files) for r, d, files in
os.walk(validation_data_dir)])
epochs = 50
```

```
batch_size = 16

if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

model = Sequential()
model.add(Conv2D(16, (5, 5), input_shape=input_shape,
padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))  # groupe 2x2 pixel

model.add(Conv2D(32, (5, 5)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (5, 5)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # mise à plat des coeffs des neurones
model.add(Dense(64))  # dense = fully connected
model.add(Activation('relu'))
model.add(Dropout(0.25))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
```

```
# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(train_data_dir,
target_size=(img_width, img_height),

batch_size=batch_size, classes=None, class_mode='categorical',
                                                    color_mode='rgb',

interpolation='bilinear')

validation_generator =
test_datagen.flow_from_directory(validation_data_dir,
target_size=(img_width, img_height),

batch_size=batch_size, classes=None, class_mode='categorical',

color_mode='rgb', interpolation='bilinear')

history = model.fit_generator(train_generator,
steps_per_epoch=nb_train_samples // batch_size, epochs=epochs,
                              validation_data=validation_generator,
                              validation_steps=nb_validation_samples
                              verbose=2)

model.save('my_model.h5')
```

Résultats

```
Found 273 images belonging to 5 classes. # images de db_train
Found 282 images belonging to 5 classes. # images de db_valid
Epoch 1/50
 - 2s - loss: 1.4562 - accuracy: 0.3268 - val_loss: 1.3435 -
val_accuracy: 0.3125
#....
Epoch 50/50
 - 2s - loss: 0.2203 - accuracy: 0.9339 - val_loss: 2.8449 -
val_accuracy: 0.7368
```

Test

Le modèle précédemment généré est ré-utilisé afin de diriger les décisions lors du traitement d'images, cela tout en mettant à jour le modèle en fonction de ces choix.

Code

```
import numpy as np
from keras.preprocessing.image import image
from keras.models import load_model
import os
import shutil
from sklearn.metrics import classification_report, confusion_matrix

def reports(y_pred, y_true, classNames):
    classification = classification_report(y_true, y_pred,
target_names=classNames)
    confusion = confusion_matrix(y_true, y_pred)
    return classification, confusion

model = load_model('my_model.h5')
img_width, img_height = 64, 64

img_dir = '../CorelDB/db_test'
resu_dir = '../result/CNN'

n_Test = sum([len(files) for r, d, files in os.walk(img_dir)])
batch_holder = np.zeros((n_Test, img_width, img_height, 3))
y_true = np.zeros(n_Test)

classNames = next(os.walk(img_dir))[1]
classNames.sort()
i = 0
for dirpath, dirnames, filenames in os.walk(img_dir):
    for imgnm in filenames:
        img = image.load_img(os.path.join(dirpath, imgnm),
target_size=(img_width, img_height))
        batch_holder[i, :] = img
        y_true[i] = int(classNames.index(os.path.relpath(dirpath,
```

```
img_dir)))
        i = i + 1

y_pred = model.predict_classes(batch_holder)
classification, confusion = reports(y_pred, y_true, classNames)

print(classification)
print(confusion)

for dirpath, dirnames, filenames in os.walk(img_dir):
    structure = os.path.join(resu_dir, os.path.relpath(dirpath,
img_dir))
    if not os.path.isdir(structure):
        os.mkdir(structure)
    else:
        print("Folder already exists")

i = 0
for dirpath, dirnames, filenames in os.walk(img_dir):
    structure = os.path.join(resu_dir, os.path.relpath(dirpath,
img_dir))
    for imgnm in filenames:
        shutil.copy(dirpath + '/' + imgnm, resu_dir + '/' +
classNames[y_pred[i]] + '/' + imgnm)
        i = i + 1
```

Résultats

```
              precision    recall  f1-score   support

       MAGEL       0.53      0.83      0.65       120
  bld_modern       0.91      0.67      0.78        95
         sc_       0.73      0.40      0.52        40
   sc_indoor       0.51      0.57      0.54        80
     wl_lion       1.00      0.10      0.18        40

    accuracy                           0.61       375
   macro avg       0.74      0.52      0.53       375
weighted avg       0.69      0.61      0.59       375

#matrice de confusion

[[100   4   1  15   0]
 [ 30  64   1   0   0]
 [  5   0  16  19   0]
 [ 31   0   3  46   0]
 [ 23   2   1  10   4]]
```

Conclusion

Pendant ce projet tout n'a pas pu être implémenté, tels que rajouter la notion de poids ou la prédiction. Cela n'a pas empêché de pouvoir d'avoir du code fonctionnel, ainsi que des résultats à comparer.

Comparatif

Parmi les deux façons d'analyser les images, il n'y a pas de meilleur choix. Chacune est adaptée à un besoin différent. Les méthodes old school sont plus adaptées à des cas spécifiques où on se concentre sur certain(s) attribut(s) de l'image, en contrepartie elles sont plus coûteuses en ressources. Celles new school permettent un traitement plus vaste et rapide mais ne n'ont pas la personnalisation proposée par le old school.