UNIVERSIDAD DE GRANADA

MÁSTER EN ELECTRÓNICA INDUSTRIAL

Procesamiento Avanzado de Imagen.

INFORME ACADÉMICO

Practica

Segmentación y Clasificación para inspección de botellas.

Presentado por:

Jorge L. Ramírez Pérez

Curso académico 2020/2021

UNIVERSIDAD
DE GRANADA

Practica

Segmentación y Clasificación para inspección de botellas.

Autor: Jorge Luis Ramirez Perez
Máster electrónica industrial, Universidad de Granada, Granada, España
Correo-e: jorgera8@correo.ugr.es

I. INTRODUCCIÓN

La aplicación más extendida del procesamiento de imagen en entornos industriales es la inspección visual de los productos que se fabrican en una cadena de producción automatizada. En esta práctica consideramos el problema de la detección automática de fallos en el envasado de líquidos. Con ella se pretende desarrollar métodos basados en técnicas de segmentación de imagen para sistemas de inspección por visión de tanto el nivel de llenado como de la colocación de tapones. Nótese que el sobrellenado de producto innecesario es una fuente de pérdida de beneficios, y los productos que no se llenan lo suficiente pueden ocasionar problemas por parte de los reguladores gubernamentales. Por otro lado, la ausencia de tapones o su colocación incorrecta en un envase de producto pueden ocasionar un desperdicio innecesario del producto y una costosa repetición del trabajo, lo que se traduce igualmente en pérdida de beneficios.

II. BASE DE DATOS DISPONIBLE:

En esta práctica se aborda el desarrollo de un sistema automático de inspección visual de una planta de embotellado de refrescos. Se facilita una base de datos de imágenes que contiene botellas correctamente envasadas, así como otras con una serie de fallos producidos en la cadena de producción [1]. La figura 1 muestra ejemplos de esta base de datos de imágenes en las que se ilustran los diferentes fallos que se producen habitualmente en este tipo de plantas automáticas de embotellamiento de bebidas. Correctamente envasadas Por debajo del nivel Por encima del nivel Sin etiqueta Etiqueta no impresa Etiqueta mal colocada Sin tapón Envase deformado Envase perdido
Figure 1. Ejemplos de imágenes de la base de datos.

Figure 1. Ejemplos de imágenes de la base de datos.

III. REALIZACION PRÁCTICA:

En esta práctica se ha de realizar el desarrollo por separado de dos scripts de Matlab para:

- Inspección automática del nivel de llenado de la botella,
- Comprobación de la correcta colocación de tapones.

3.1. Desarrollar un método automático de comprobación del nivel de llenado La carpeta “Normal” contiene 30 imágenes de botellas con un correcto nivel de llenado. Las carpetas “1-UnderFilled” y “2-OverFilled” agrupan 10 botellas en cada una con defecto de líquido y con sobrellenado, respectivamente. Desarrolle un método basado en técnicas de segmentación de imagen para detectar de forma automática el correcto/incorrecto nivel de llenado.

3.2. Desarrollar un método automático de inspección de tapones La carpeta “Normal” contiene 30 imágenes de botellas con una correcta colocación del tapón de la botella. La carpeta “6-CapMissing” agrupa 10 botellas en las que no se ha colocado el tapón a la botella. Desarrolle un método basado en técnicas de segmentación de imagen para detectar de forma automática la presencia/ausencia del tapón en la botella.

IV. ALGORITMOS

En el desarrollo de los algoritmos para detectar automáticamente la comprobación del nivel de llenado, sobre llenado o falta de llenado (OverFilled, UnderFilled) y tambien para ver si falta la tapa o esta normal. Se nos pidió realizar dos puntos por separado, pero al realizar Redes Neuronales convolucionales he decido crear las 4 clases en un mismo código:

Clases:

- UnderFilled
- OverFilled
- CapMissing
- Normal

Para comenzar se aplicará una segmentación usando el método de superpíxeles el cual nos brinda una segmentación incrementando los pixeles de la imagen y así poder sacar formas, modelos y las características necesarias para clasificar el objeto necesario.

El código realizado es el siguiente:

```
%% Lectura secuencial de las imágenes de una determinada clase.
% leemos de los diferentes paths para poder segmentar todas las imagenes de
% cada carpeta.

% path= './1-UnderFilled';
%path= './2-OverFilled';
%path= './6-CapMissing';
path= './All';
%path= './Normal';


files= dir([path '/*.jpg' ]);
Nfiles= size(files,1);

%leemos todas las imagenes y le aplicamos la segmentación por super
%pixeles, para posteriormente mediante una red neuronal en python
%clasificar estas imagenes.
```

```
for file=1:Nfiles
 f = imread([path '/' files(file).name]);
 %imshow(f);
 pause(0.01);
 %figure;
%imshow(f);
whos f;

    fLab= rgb2lab(f);

    [L,N]= superpixels(fLab,6500);

%     figure;
    BW= boundarymask(L);
%     imshow(imoverlay(f,BW,'cyan'),'InitialMagnification',67);

    %Creamos una lista de los pixeles en cada región.
    pixelIdxList = label2idx(L);

    meanColor = zeros(N,3);

    [m,n] = size(L);
    for  i = 1:N
        meanColor(i,1) = mean(fLab(pixelIdxList{i}));
        meanColor(i,2) = mean(fLab(pixelIdxList{i}+m*n));
        meanColor(i,3) = mean(fLab(pixelIdxList{i}+2*m*n));
    end

    numColors= 3;
    [idx,cmap]= kmeans(meanColor,numColors,'replicates',2);
    cmap= lab2rgb(cmap);
    Lout= zeros(size(f,1),size(f,2));Lout
    for i=1:N
        Lout(pixelIdxList{i}) = idx(i);
    end
%     figure;
%     imshow(label2rgb(Lout)) ;
%     figure;
%     imshow(Lout,cmap);

    %imwrite(label2rgb(Lout),sprintf('IMAGEN_0%d.jpg',file))
    %imwrite(label2rgb(Lout),sprintf('CapMissing_0%d.jpg',file))
    %imwrite(Lout,cmap,sprintf('UnderFilled_0%d.jpg',file))
    %imwrite(Lout,cmap,sprintf('OverFilled_0%d.jpg',file))
    %imwrite(Lout,cmap,sprintf('CapMissing_0%d.jpg',file))
    imwrite(Lout,cmap,sprintf('All_0%d.jpg',file))
    %imwrite(Lout,cmap,sprintf('Normal_0%d.jpg',file))
end
```

Con este código leemos todas las imágenes de cada carpeta; Underfilled, Overfilled, CapMissing y Normal, para generar una segmentación con superpixels y posteriormente una segmentación de color para los vecinos con la función de Kmeans (‘Replicates’) y asignamos un numero de colores =3, el cual es más qué necesario para poder diferenciar la tapa, el líquido, el fondo y la etiqueta. Una opción para identificar la etiqueta podría ser escoger cantidad de colores 4 para diferenciar la etiqueta de la tapa con un filtro.

Imágenes resultantes después de la segmentación:

Imagen segmentada por superpixels Imagen Original

Aplicamos la segmentación a todas las imágenes para llevarlas a nuestra red neuronal desarrollada en Python, de esta forma es mucho más fácil para la red poder aprender a diferenciar las clases y obtener un resultado más preciso.

Red Neuronal CNN:

Seleccionamos el Path para leer las imágenes las cuales guarde también como los nombres originales la mitad en el entrenamiento y la otra mitad en la validación. Para el entrenamiento de esta red neuronal es importante saber que todas las imágenes que se tiene para ella tienen el defecto siempre en la botella del medio, es por esta razón que la red neuronal no aprender a clasificar si el defecto esta en una de las botellas vecinas. De tener una mayor cantidad de fotos en la base de datos, la red sería mucho mas precisa y potente para poder clasificar mejor cada clase.

TrainJorgeRamirez.py:

```python
import sys
import os
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras import optimizers
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.python.keras.layers import  Convolution2D, MaxPooling2D
from tensorflow.python.keras import backend as K
import tensorflow as tf


K.clear_session()


# ingresamos el path para los valores de entrenamiento y validacion
data_entrenamiento = './data/entrenamiento'
data_validacion = './data/validacion'

# En este caso en particular debido a la poca cantidad de imagenes para entrenamiento
# Decidí usar un 50%-50% de imagenes para entrenar y validar
# 5 imagenes para cada uno

"""
Parametros
"""
epocas=20 # numero de veces de iteraciones de todos los datos durante el entrenmaiento
longitud, altura = 150, 150 # tamaño de procesamiento de imagenes
batch_size = 1 # numero de imagenes que se enviara en cada una de los pasos
pasos = 15 # Es importante que el numero de pasos no exceda la cantidad de datos.
validation_steps = 150  # numero de veces que se procesa la información en cada epoca
# debido a la poca cantidad de imagenes podemos escoger un numero de pasos de 150
filtrosConv1 = 32
filtrosConv2 = 64
tamano_filtro1 = (3, 3)# dos filtros para cada capa
tamano_filtro2 = (2, 2)
tamano_pool = (2, 2) #el tamaño del pool
clases = 4
lr = 0.0004 # rango de aprendizaje  que va actuando con la funcion de activacion


##Preparamos nuestras imagenes

entrenamiento_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

entrenamiento_generador = entrenamiento_datagen.flow_from_directory(
    data_entrenamiento,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')

```

```
validacion_generador = test_datagen.flow_from_directory(
    data_validacion,
    target_size=(altura, longitud),
    batch_size=batch_size,
    class_mode='categorical')

# Creamos Red CNN

cnn = Sequential()

# capa 1
cnn.add(Convolution2D(filtrosConv1, tamano_filtro1, padding ="same", input_shape=(longitud, altura, 3), activation='relu'))
#pooling 1
cnn.add(MaxPooling2D(pool_size=tamano_pool))
# capa 2
cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, padding ="same"))
#pooling 2
cnn.add(MaxPooling2D(pool_size=tamano_pool))

# aplanamos los datos
cnn.add(Flatten())
#denisdad de neuronas
cnn.add(Dense(256, activation='relu'))
#ajuste de paso por las neuronas
cnn.add(Dropout(0.5))
# El que nos ayudara a identificar las diferentes clases
cnn.add(Dense(clases, activation='softmax'))

cnn.compile(loss='categorical_crossentropy',
            optimizer=tf.keras.optimizers.Adam(lr=lr), # me generaba error la libreria de ADAM como tensor.python.keras.optimizer import ADAM
            metrics=['accuracy'])




cnn.fit_generator(
    entrenamiento_generador,
    steps_per_epoch=pasos,
    epochs=epocas,
    validation_data=validacion_generador,
    validation_steps=validation_steps)

target_dir = './modelo/'  #guardamos la estructura
if not os.path.exists(target_dir):
  os.mkdir(target_dir)
cnn.save('./modelo/modelo.h5')#guardamos los pesos del modelo
cnn.save_weights('./modelo/pesos.h5')
```

Predictjorgeramirez.py:

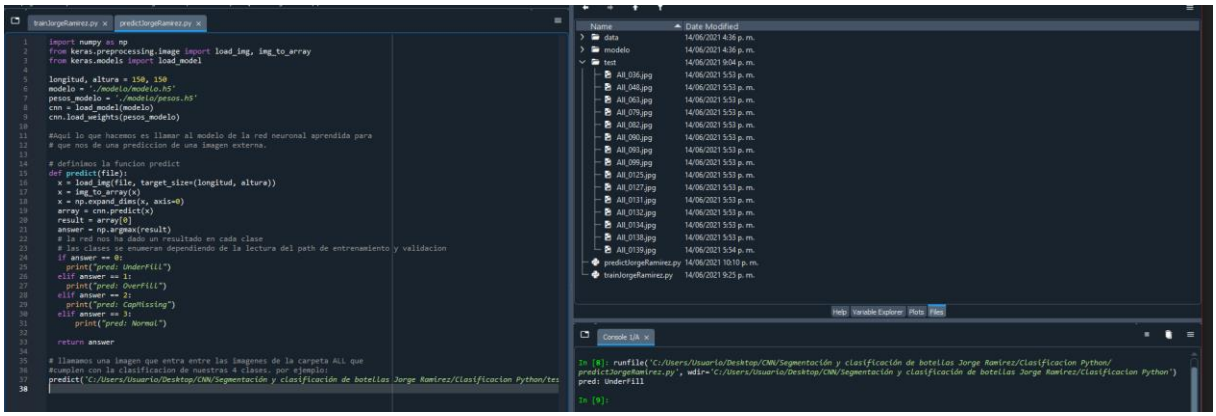
```
import numpy as np
from keras.preprocessing.image import load_img, img_to_array
from keras.models import load_model

longitud, altura = 150, 150
modelo = './modelo/modelo.h5'
pesos_modelo = './modelo/pesos.h5'
cnn = load_model(modelo)
cnn.load_weights(pesos_modelo)

#Aqui lo que hacemos es llamar al modelo de la red neuronal aprendida para
# que nos de una prediccion de una imagen externa.

# definimos la funcion predict
def predict(file):
  x = load_img(file, target_size=(longitud, altura))
  x = img_to_array(x)
  x = np.expand_dims(x, axis=0)
  array = cnn.predict(x)
  result = array[0]
  answer = np.argmax(result)
  # la red nos ha dado un resultado en cada clase
  # las clases se enumeran dependiendo de la lectura del path de entrenamiento y validacion
  if answer == 0:
    print("pred: UnderFill")
  elif answer == 1:
    print("pred: OverFill")
  elif answer == 2:
    print("pred: CapMissing")
  elif answer == 3:
      print("pred: Normal")

  return answer

# llamamos una imagen que entra entre las imagenes de la carpeta ALL que
#cumplen con la clasificacion de nuestras 4 clases. por ejemplo:
predict('C:/Users/Usuario/Desktop/CNN/CNN desde cero/test/All_090.jpg')
```

Entrenamiento:

```
Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-06-14 22:06:56.873269: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1261] Device interconnect StreamExecutor with strength 1 edge matrix:
2021-06-14 22:06:56.873281: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1267]
2021-06-14 22:06:56.873294: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_enable_xla_devices not set
2021-06-14 22:06:57.229298: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
Epoch 1/20
15/15 [==============================] - ETA: 0s - loss: 8.0685 - accuracy: 0.4666WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 150 batches). You may need to use the repeat() function when building your dataset.
15/15 [==============================] - 4s 202ms/step - loss: 8.0579 - accuracy: 0.4666 - val_loss: 4.8089 - val_accuracy: 0.5000
Epoch 2/20
15/15 [==============================] - 2s 107ms/step - loss: 5.9410 - accuracy: 0.3673
Epoch 3/20
15/15 [==============================] - 2s 101ms/step - loss: 5.4586 - accuracy: 0.1098
Epoch 4/20
15/15 [==============================] - 2s 109ms/step - loss: 2.6201 - accuracy: 0.3946
Epoch 5/20
15/15 [==============================] - 1s 96ms/step - loss: 2.3304 - accuracy: 0.4941
Epoch 6/20
15/15 [==============================] - 2s 124ms/step - loss: 1.5794 - accuracy: 0.6550
Epoch 7/20
15/15 [==============================] - 2s 111ms/step - loss: 1.5844 - accuracy: 0.5222
Epoch 8/20
15/15 [==============================] - 2s 103ms/step - loss: 0.7960 - accuracy: 0.8535
Epoch 9/20
15/15 [==============================] - 2s 107ms/step - loss: 1.4322 - accuracy: 0.4789
Epoch 10/20
15/15 [==============================] - 2s 115ms/step - loss: 0.4742 - accuracy: 0.8344
Epoch 11/20
15/15 [==============================] - 2s 101ms/step - loss: 0.2589 - accuracy: 0.8981
Epoch 12/20
15/15 [==============================] - 1s 94ms/step - loss: 0.4259 - accuracy: 0.9525
Epoch 13/20
15/15 [==============================] - 1s 98ms/step - loss: 0.4614 - accuracy: 0.8422
Epoch 14/20
15/15 [==============================] - 2s 122ms/step - loss: 0.5334 - accuracy: 0.8736
Epoch 15/20
15/15 [==============================] - 2s 105ms/step - loss: 0.5081 - accuracy: 0.6035
Epoch 16/20
15/15 [==============================] - 2s 112ms/step - loss: 0.1895 - accuracy: 0.9653
Epoch 17/20
15/15 [==============================] - 2s 104ms/step - loss: 0.4615 - accuracy: 0.7769
Epoch 18/20
15/15 [==============================] - 2s 103ms/step - loss: 0.4854 - accuracy: 0.8822
Epoch 19/20
15/15 [==============================] - 1s 98ms/step - loss: 0.1543 - accuracy: 1.0000
Epoch 20/20
15/15 [==============================] - 2s 101ms/step - loss: 0.0120 - accuracy: 1.0000
```

Resultado:

Imagen test All_090.jpg



runfile('C:/Users/Usuario/Desktop/CNN/Segmentación y clasificación de botellas Jorge Ramirez/Clasificacion Python/predictJorgeRamirez.py', wdir='C:/Users/Usuario/Desktop/CNN/Segmentación y clasificación de botellas Jorge Ramirez/Clasificacion Python')

pred: UnderFill

Predicción correcta.