**Codigo generador de imágenes**

from PIL import Image

from random import randrange

from numpy import random

lista = ['imagenes/placa0.jpg', 'imagenes/placa1.jpg', 'imagenes/placa2.jpg']

lista2 = ['imagenes/comp.png', 'imagenes/comp0.png', 'imagenes/comp1.png', 'imagenes/comp2.png']

lista3 = ['imagenes/comp.jpg', 'imagenes/comp0.jpg', 'imagenes/comp1.jpg', 'imagenes/comp2.jpg']

x = 0

num\_generar = 80

for i in range(num\_generar):

#imagen de fondo

fondo = Image.open(random.choice(lista))

#imagen tipo .png para colocar sobre el fondo

objeto = Image.open(random.choice(lista2)).convert("RGBA")

objeto = objeto.resize((objeto.width//3,objeto.height//3))

objeto = objeto.rotate(randrange(180))

#posicion random dentro de la imagen

pos = ((fondo.width - randrange(fondo.width-objeto.width)), (fondo.height - randrange(fondo.height-objeto.height)))

img\_nueva = fondo.copy()

img\_nueva.paste(objeto, pos, objeto)

name = ('Imagenes\_generadas/' + str(i) + '.jpg')

img\_nueva.save(name)

for i in lista3:

imag = Image.open(i)

imag\_rotada = imag.rotate(randrange(180))

x = x + 1

imag\_rotada.save('Imagenes\_generadas/mcc\_rotada' + str(x) + '.jpg')

**Codigo para los archivos cvs**

import os

import glob

import pandas as pd

import xml.etree.ElementTree as ET

def xml\_to\_csv(path):

xml\_list = []

for xml\_file in glob.glob(path + '/\*.xml'):

tree = ET.parse(xml\_file)

root = tree.getroot()

for member in root.findall('object'):

value = (root.find('filename').text,

int(root.find('size')[0].text),

int(root.find('size')[1].text),

member[0].text,

int(member[4][0].text),

int(member[4][1].text),

int(member[4][2].text),

int(member[4][3].text)

)

xml\_list.append(value)

column\_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']

xml\_df = pd.DataFrame(xml\_list, columns=column\_name)

return xml\_df

def main():

for directory in ['train','test']:

image\_path = os.path.join(os.getcwd(), ' Imagenes\_generadas/'.format(directory))

xml\_df = xml\_to\_csv(image\_path)

xml\_df.to\_csv('data/{}\_labels.csv'.format(directory), index=None)

print('Successfully converted xml to csv.')

main()

**Codigo de los archivos tfrecord**

"""

Usage:

# From tensorflow/models/

# Create train data:

python generate\_tfrecord.py --csv\_input=data/train\_labels.csv --output\_path=train.record -- image\_dir=Imagenes\_generadas/

# Create test data:

python generate\_tfrecord.py --csv\_input=data/test\_labels.csv --output\_path=test.record -- image\_dir=Imagenes\_generadas/

"""

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

from \_\_future\_\_ import absolute\_import

import os

import io

import pandas as pd

import tensorflow as tf

from PIL import Image

from object\_detection.utils import dataset\_util

from collections import namedtuple, OrderedDict

flags = tf.app.flags

flags.DEFINE\_string('csv\_input', '', 'Path to the CSV input')

flags.DEFINE\_string('output\_path', '', 'Path to output TFRecord')

flags.DEFINE\_string('image\_dir', '', 'Path to images')

FLAGS = flags.FLAGS

# TO-DO replace this with label map

def class\_text\_to\_int(row\_label):

if row\_label == 'componente\_T':

return 1

else:

None

def split(df, group):

data = namedtuple('data', ['filename', 'object'])

gb = df.groupby(group)

return [data(filename, gb.get\_group(x)) for filename, x in zip(gb.groups.keys(), gb.groups)]

def create\_tf\_example(group, path):

with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:

encoded\_jpg = fid.read()

encoded\_jpg\_io = io.BytesIO(encoded\_jpg)

image = Image.open(encoded\_jpg\_io)

width, height = image.size

filename = group.filename.encode('utf8')

image\_format = b'jpg'

xmins = []

xmaxs = []

ymins = []

ymaxs = []

classes\_text = []

classes = []

for index, row in group.object.iterrows():

xmins.append(row['xmin'] / width)

xmaxs.append(row['xmax'] / width)

ymins.append(row['ymin'] / height)

ymaxs.append(row['ymax'] / height)

classes\_text.append(row['class'].encode('utf8'))

classes.append(class\_text\_to\_int(row['class']))

tf\_example = tf.train.Example(features=tf.train.Features(feature={

'image/height': dataset\_util.int64\_feature(height),

'image/width': dataset\_util.int64\_feature(width),

'image/filename': dataset\_util.bytes\_feature(filename),

'image/source\_id': dataset\_util.bytes\_feature(filename),

'image/encoded': dataset\_util.bytes\_feature(encoded\_jpg),

'image/format': dataset\_util.bytes\_feature(image\_format),

'image/object/bbox/xmin': dataset\_util.float\_list\_feature(xmins),

'image/object/bbox/xmax': dataset\_util.float\_list\_feature(xmaxs),

'image/object/bbox/ymin': dataset\_util.float\_list\_feature(ymins),

'image/object/bbox/ymax': dataset\_util.float\_list\_feature(ymaxs),

'image/object/class/text': dataset\_util.bytes\_list\_feature(classes\_text),

'image/object/class/label': dataset\_util.int64\_list\_feature(classes),

}))

return tf\_example

def main(\_):

writer = tf.python\_io.TFRecordWriter(FLAGS.output\_path)

path = os.path.join(FLAGS.image\_dir)

examples = pd.read\_csv(FLAGS.csv\_input)

grouped = split(examples, 'filename')

for group in grouped:

tf\_example = create\_tf\_example(group, path)

writer.write(tf\_example.SerializeToString())

writer.close()

output\_path = os.path.join(os.getcwd(), FLAGS.output\_path)

print('Successfully created the TFRecords: {}'.format(output\_path))

if \_\_name\_\_ == '\_\_main\_\_':

tf.app.run()

**Codigo De Entrenamiento**

# SSD with Inception v2 configuration for MSCOCO Dataset.

# Users should configure the fine\_tune\_checkpoint field in the train config as

# well as the label\_map\_path and input\_path fields in the train\_input\_reader and

# eval\_input\_reader. Search for "PATH\_TO\_BE\_CONFIGURED" to find the fields that

# should be configured.

model {

ssd {

num\_classes: 1

box\_coder {

faster\_rcnn\_box\_coder {

y\_scale: 10.0

x\_scale: 10.0

height\_scale: 5.0

width\_scale: 5.0

}

}

matcher {

argmax\_matcher {

matched\_threshold: 0.5

unmatched\_threshold: 0.5

ignore\_thresholds: false

negatives\_lower\_than\_unmatched: true

force\_match\_for\_each\_row: true

}

}

similarity\_calculator {

iou\_similarity {

}

}

anchor\_generator {

ssd\_anchor\_generator {

num\_layers: 6

min\_scale: 0.2

max\_scale: 0.95

aspect\_ratios: 1.0

aspect\_ratios: 2.0

aspect\_ratios: 0.5

aspect\_ratios: 3.0

aspect\_ratios: 0.3333

reduce\_boxes\_in\_lowest\_layer: true

}

}

image\_resizer {

fixed\_shape\_resizer {

height: 300

width: 300

}

}

box\_predictor {

convolutional\_box\_predictor {

min\_depth: 0

max\_depth: 0

num\_layers\_before\_predictor: 0

use\_dropout: false

dropout\_keep\_probability: 0.8

kernel\_size: 3

box\_code\_size: 4

apply\_sigmoid\_to\_scores: false

conv\_hyperparams {

activation: RELU\_6,

regularizer {

l2\_regularizer {

weight: 0.00004

}

}

initializer {

truncated\_normal\_initializer {

stddev: 0.03

mean: 0.0

}

}

}

}

}

feature\_extractor {

type: 'ssd\_inception\_v2'

min\_depth: 16

depth\_multiplier: 1.0

conv\_hyperparams {

activation: RELU\_6,

regularizer {

l2\_regularizer {

weight: 0.00004

}

}

initializer {

truncated\_normal\_initializer {

stddev: 0.03

mean: 0.0

}

}

batch\_norm {

train: true,

scale: true,

center: true,

decay: 0.9997,

epsilon: 0.001,

}

}

}

loss {

classification\_loss {

weighted\_sigmoid {

}

}

localization\_loss {

weighted\_smooth\_l1 {

}

}

hard\_example\_miner {

num\_hard\_examples: 3000

iou\_threshold: 0.99

loss\_type: CLASSIFICATION

max\_negatives\_per\_positive: 3

min\_negatives\_per\_image: 0

}

classification\_weight: 1.0

localization\_weight: 1.0

}

normalize\_loss\_by\_num\_matches: true

post\_processing {

batch\_non\_max\_suppression {

score\_threshold: 1e-8

iou\_threshold: 0.6

max\_detections\_per\_class: 100

max\_total\_detections: 100

}

score\_converter: SIGMOID

}

}

}

train\_config: {

batch\_size: 24

optimizer {

rms\_prop\_optimizer: {

learning\_rate: {

exponential\_decay\_learning\_rate {

initial\_learning\_rate: 0.004

decay\_steps: 800720

decay\_factor: 0.95

}

}

momentum\_optimizer\_value: 0.9

decay: 0.9

epsilon: 1.0

}

}

fine\_tune\_checkpoint: "data/model.ckpt.index"

from\_detection\_checkpoint: true

# Note: The below line limits the training process to 200K steps, which we

# empirically found to be sufficient enough to train the pets dataset. This

# effectively bypasses the learning rate schedule (the learning rate will

# never decay). Remove the below line to train indefinitely.

num\_steps: 200000

data\_augmentation\_options {

random\_horizontal\_flip {

}

}

data\_augmentation\_options {

ssd\_random\_crop {

}

}

}

train\_input\_reader: {

tf\_record\_input\_reader {

input\_path: "data/train.record"

}

label\_map\_path: "data/label\_map.pbtxt"

}

eval\_config: {

num\_examples: 8000

# Note: The below line limits the evaluation process to 10 evaluations.

# Remove the below line to evaluate indefinitely.

max\_evals: 10

}

eval\_input\_reader: {

tf\_record\_input\_reader {

input\_path: "data/test.record"

}

label\_map\_path: "data/label\_map.pbtxt"

shuffle: false

num\_readers: 1

}