Object detection using Mask R-CNN on a custom dataset

Renu Khandelwal
Nov 28, 2019 · 6 min read ★

In this article we will implement Mask R-CNN for detecting objects from a custom dataset

Prerequisites:

Computer vision : A journey from CNN to Mask R-CC and YOLO Part 1

Computer vision : A journey from CNN to Mask R-CNN and YOLO Part 2

Instance segmentation using Mask R-CNN

Transfer Learning

Transfer Learning using ResNet50

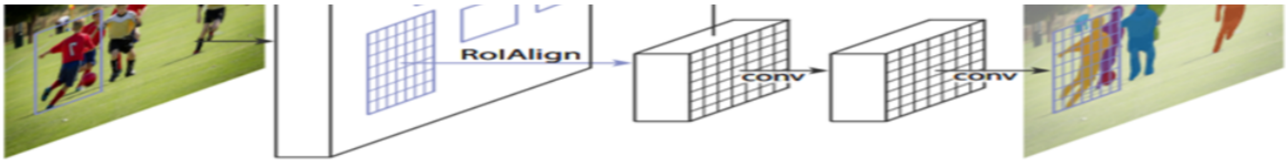
Data set

Kangaroo data set is used in the article

Mask R-CNN

Mask R-CNN is a deep neural network for instance segmentation. The model is divided into two parts

- **Region proposal network (RPN) to proposes candidate object bounding boxes.**
- **Binary mask classifier to generate mask for every class**



Mask R-CNN have a branch for classification and bounding box regression. It uses

- ResNet101 architecture to extract features from image.
- Region Proposal Network(RPN) to generate Region of Interests(RoI)

Transfer learning using Mask R-CNN Code in keras

For this we use **MatterPort Mask R-CNN.**

Step 1: Clone the Mask R-CNN repository

```
git clone https://github.com/matterport/Mask_RCNN.git
cd Mask_RCNN
$ python setup.py install
```

Step 2: Download the pre-trained weights for COCO model from MatterPort.

Place the file in the Mask_RCNN folder with name “mask_rcnn_coco.h5”

Step 3: Import the required libraries

```
from mrcnn.config import Config
from mrcnn import model as modellib
from mrcnn import visualize
import mrcnn
from mrcnn.utils import Dataset
from mrcnn.model import MaskRCNN

import numpy as np
from numpy import zeros
from numpy import asarray
import colorsys
import argparse
import imutils
import random
import cv2
import os
import time
```

```
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from keras.models import load_model

%matplotlib inline

from os import listdir
from xml.etree import ElementTree
```

Step 4: We Create a ***myMaskRCNNConfig*** class for training on the Kangaroo dataset**.** It is derived from the base ***Mask R-CNN Config*** class and overrides some values.

```
class myMaskRCNNConfig(Config):
    # give the configuration a recognizable name
    NAME = "MaskRCNN_config"

    # set the number of GPUs to use along with the number of images
    # per GPU
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

    # number of classes (we would normally add +1 for the
background)
     # kangaroo + BG
    NUM_CLASSES = 1+1

    # Number of training steps per epoch
    STEPS_PER_EPOCH = 131

    # Learning rate
    LEARNING_RATE=0.006

    # Skip detections with < 90% confidence
    DETECTION_MIN_CONFIDENCE = 0.9

    # setting Max ground truth instances
    MAX_GT_INSTANCES=10
```

Step 5: Create an instance of the *myMaskRCNNConfig* class

```
config = myMaskRCNNConfig()
```

Let’s display all the config values.

```
config.display()
```

```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.9
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 1
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  1024
IMAGE_META_SIZE                14
IMAGE_MIN_DIM                  800
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [1024 1024    3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.006
LOSS_WEIGHTS                   {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0,
'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               10
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           MaskRCNN_config
NUM_CLASSES                    2
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
PRE_NMS_LIMIT                  6000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE              1
RPN_BBOX_STD_DEV               [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD              0.7
RPN_TRAIN_ANCHORS_PER_IMAGE    256
STEPS_PER_EPOCH                131
TOP_DOWN_PYRAMID_SIZE          256
TRAIN_BN                       False
TRAIN_ROIS_PER_IMAGE           200
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```

Step 6: Build the custom kangaroo data set.

Dataset class provides a consistent way to work with any dataset. We will create our new datasets for kangaroo dataset to train without having to change the code of the model.

Dataset class also supports loading multiple data sets at the same time,. This is very helpful when the you want to detect different objects and they are all not available in one data set.

In ***load_dataset*** method, we iterate through all the files in the image and annotations folders to add the class, images and annotations to create the dataset using ***add_class*** and ***add_image*** methods.

extract_boxes method extracts each of the bounding box from the annotation file. Annotation files are xml files using pascal VOC format. It returns the box, it's height and width

load_mask method generates the masks for every object in the image. It returns one mask per instance and class ids, a 1D array of class id for the instance masks

image_reference method returns the path of the image

```
class KangarooDataset(Dataset):
    # load the dataset definitions
    def load_dataset(self, dataset_dir, is_train=True):

        # Add classes. We have only one class to add.
        self.add_class("dataset", 1, "kangaroo")

        # define data locations for images and annotations
        images_dir = dataset_dir + '\\images\\'
        annotations_dir = dataset_dir + '\\annots\\'

        # Iterate through all files in the folder to
        #add class, images and annotaions
        for filename in listdir(images_dir):

            # extract image id
            image_id = filename[:-4]

            # skip bad images
            if image_id in ['00090']:
                continue
            # skip all images after 150 if we are building the train
set
            if is_train and int(image_id) >= 150:
                continue
            # skip all images before 150 if we are building the
test/val set
            if not is_train and int(image_id) < 150:
                continue

            # setting image file
            img_path = images_dir + filename

            # setting annotations file
            ann_path = annotations_dir + image_id + '.xml'

            # adding images and annotations to dataset
            self.add_image('dataset', image_id=image_id,
path=img_path, annotation=ann_path)

# extract bounding boxes from an annotation file
    def extract_boxes(self, filename):
```

```
        # load and parse the file
        tree = ElementTree.parse(filename)
        # get the root of the document
        root = tree.getroot()
        # extract each bounding box
        boxes = list()
        for box in root.findall('.//bndbox'):
            xmin = int(box.find('xmin').text)
            ymin = int(box.find('ymin').text)
            xmax = int(box.find('xmax').text)
            ymax = int(box.find('ymax').text)
            coors = [xmin, ymin, xmax, ymax]
            boxes.append(coors)

        # extract image dimensions
        width = int(root.find('.//size/width').text)
        height = int(root.find('.//size/height').text)
        return boxes, width, height

# load the masks for an image
    """Generate instance masks for an image.
       Returns:
        masks: A bool array of shape [height, width, instance count]
with
            one mask per instance.
        class_ids: a 1D array of class IDs of the instance masks.
     """
    def load_mask(self, image_id):
        # get details of image
        info = self.image_info[image_id]

        # define anntation  file location
        path = info['annotation']

        # load XML
        boxes, w, h = self.extract_boxes(path)

        # create one array for all masks, each on a different
channel
        masks = zeros([h, w, len(boxes)], dtype='uint8')

        # create masks
        class_ids = list()
        for i in range(len(boxes)):
            box = boxes[i]
            row_s, row_e = box[1], box[3]
            col_s, col_e = box[0], box[2]
            masks[row_s:row_e, col_s:col_e, i] = 1
            class_ids.append(self.class_names.index('kangaroo'))
        return masks, asarray(class_ids, dtype='int32')

# load an image reference
     """Return the path of the image."""
    def image_reference(self, image_id):
        info = self.image_info[image_id]
        print(info)
        return info['path']
```

Step 7: Prepare the train and test set

```
# prepare train set
train_set = KangarooDataset()
train_set.load_dataset('..\\Kangaroo\\kangaroo-master\\kangaroo-master', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))

# prepare test/val set
test_set = KangarooDataset()
test_set.load_dataset('..\\Kangaroo\\kangaroo-master\\kangaroo-master', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))
```

Step 8 :Initialize Mask R-CNN model for *"training"* using the Config instance that we created

```
print("Loading Mask R-CNN model...")
model = modellib.MaskRCNN(mode="training", config=config, model_dir='./')
```

Step 9: Load the pre-trained weights for the Mask R-CNN from COCO data set excluding the last few layers

We exclude the last few layers from training for ResNet101. Excluding the last layers is to match the number of classes in the new data set.

```
#load the weights for COCO
model.load_weights('.\\Mask_RCNN\\mask_rcnn_coco.h5',
                   by_name=True,
                   exclude=["mrcnn_class_logits", "mrcnn_bbox_fc", "mrcnn_bbox", "mrcnn_mask"])
```

Step 10: Train the heads with higher learning rate to speed up the learning

We can increase the speed of learning for head layers by increasing the learning rate

Also we can increase the epochs to anywhere from 100–500 and see the difference in the accuracy of the object detection. I have used only 5 epochs as I trained it on a CPU.

```
## train heads with higher lr to speedup the learning
model.train(train_set, test_set,
learning_rate=2*config.LEARNING_RATE, epochs=5, layers='heads')

history = model.keras_model.history.history
```

```
Starting at epoch 0. LR=0.006

Checkpoint Path: ./maskrcnn_config20191127T0912\mask_rcnn_maskrcnn_config_{epoch:04d}.h5
Selecting layers to train
fpn_c5p5               (Conv2D)
fpn_c4p4               (Conv2D)
fpn_c3p3               (Conv2D)
fpn_c2p2               (Conv2D)
fpn_p5                 (Conv2D)
fpn_p2                 (Conv2D)
fpn_p3                 (Conv2D)
fpn_p4                 (Conv2D)
In model:  rpn_model
    rpn_conv_shared        (Conv2D)
    rpn_class_raw          (Conv2D)
    rpn_bbox_pred          (Conv2D)
mrcnn_mask_conv1       (TimeDistributed)
mrcnn_mask_bn1         (TimeDistributed)
mrcnn_mask_conv2       (TimeDistributed)
mrcnn_mask_bn2         (TimeDistributed)
mrcnn_class_conv1      (TimeDistributed)
mrcnn_class_bn1        (TimeDistributed)
mrcnn_mask_conv3       (TimeDistributed)
mrcnn_mask_bn3         (TimeDistributed)
mrcnn_class_conv2      (TimeDistributed)
mrcnn_class_bn2        (TimeDistributed)
mrcnn_mask_conv4       (TimeDistributed)
mrcnn_mask_bn4         (TimeDistributed)
mrcnn_bbox_fc          (TimeDistributed)
mrcnn_mask_deconv      (TimeDistributed)
mrcnn_class_logits     (TimeDistributed)
mrcnn_mask             (TimeDistributed)
WARNING:tensorflow:From C:\Users\khandelwalr\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\ops\math_op
s.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```

Step 11: Save the trained weights for custom data set

```
import time

model_path = '..\\Kangaroo\\kangaroo-master\\kangaroo-
master\\mask_rcnn_'  + '.' + str(time.time()) + '.h5'

model.keras_model.save_weights(model_path)
```

Step 12: Detecting objects in the image with masks and bounding box from the trained model

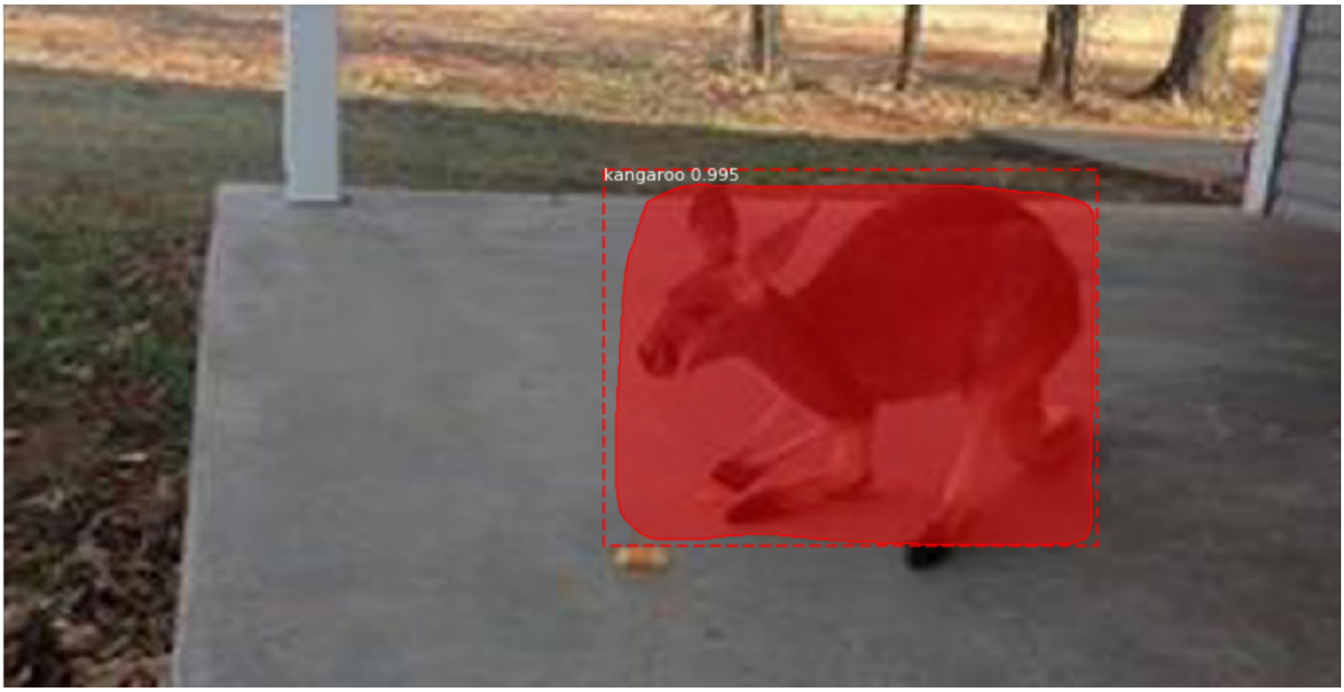
Create the model in the inference mode. Load the weights for the model from the data set that we trained the model on.

Load the image that we want to detect the bounding boxes, classes and confidence percentage

```
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array

#Loading the model in the inference mode
model = modellib.MaskRCNN(mode="inference", config=config,
model_dir='./')

# loading the trained weights o the custom dataset
model.load_weights(model_path, by_name=True)

img = load_img("..\\Kangaroo\\kangaroo-master\\kangaroo-
master\\images\\00042.jpg")
img = img_to_array(img)

# detecting objects in the image
result= model.detect([img])
```

Finally displaying the results

```
image_id = 20
image, image_meta, gt_class_id, gt_bbox, gt_mask =
modellib.load_image_gt(test_set, config, image_id,
use_mini_mask=False)

info = test_set.image_info[image_id]
print("image ID: {}.{} ({}) {}".format(info["source"], info["id"],
image_id,

test_set.image_reference(image_id)))

# Run object detection
results = model.detect([image], verbose=1)

# Display results

r = results[0]
visualize.display_instances(image, r['rois'], r['masks'],
r['class_ids'],
                            test_set.class_names, r['scores'],
                            title="Predictions")
```



References:

matterport/Mask_RCNN

This is an implementation of Mask R-CNN on Python 3, Keras, and TensorFlow. The model generates bounding boxes and…

github.com

https://github.com/arshren/Mask_RCNN/blob/master/Transfer%20Learning%20Mask%20RCNN-Custom%20dataset.ipynb

Splash of Color: Instance Segmentation with Mask R-CNN and TensorFlow

Explained by building a color splash filter

engineering.matterport.com

https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/

Mask R Cnn Keras Object Detection Deep Learning Computer Vision