Mask R-CNN - Train on Shapes Dataset

This notebook shows how to train Mask R-CNN on your own dataset. To keep things simple we use a synthetic dataset of shapes (squares, triangles, and circles) which enables fast training. You'd still need a GPU, though, because the network backbone is a Resnet101, which would be too slow to train on a CPU. On a GPU, you can start to get okay-ish results in a few minutes, and good results in less than an hour.

The code of the *Shapes* dataset is included below. It generates images on the fly, so it doesn't require downloading any data. And it can generate images of any size, so we pick a small image size to train faster.

In [1]:

```
import os
import sys
import random
import math
import re
import time
import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt

from config import Config
import utils
import model as modellib
import visualize
from model import log

%matplotlib inline

# Root directory of the project
ROOT_DIR = os.getcwd()

# Directory to save logs and trained model
MODEL_DIR = os.path.join(ROOT_DIR, "logs")

# Local path to trained weights file
COCO_MODEL_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")
# Download COCO trained weights from Releases if needed
if not os.path.exists(COCO_MODEL_PATH):
    utils.download_trained_weights(COCO_MODEL_PATH)
```

```
/usr/local/lib/python3.5/dist-packages/h5py/__init__.py:36: FutureWa
rning: Conversion of the second argument of issubdtype from `float`
to `np.floating` is deprecated. In future, it will be treated as `n
p.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

Configurations

In [2]:

```
class BarrierConfig(Config):
    """Configuration for training on the toy shapes dataset.
    Derives from the base Config class and overrides values specific
    to the toy shapes dataset.
    """
    # Give the configuration a recognizable name
    NAME = "bar"

    # Train on 1 GPU and 8 images per GPU. We can put multiple images on each
    # GPU because the images are small. Batch size is 8 (GPUs * images/GPU).
    IMAGES_PER_GPU = 2

    # Number of classes (including background)
    NUM_CLASSES = 1 + 1  # background + 3 shapes

config = BarrierConfig()
config.display()
```

```
Configurations:
BACKBONE_SHAPES                [[256 256]
 [128 128]
 [ 64  64]
 [ 32  32]
 [ 16  16]]
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     2
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.7
DETECTION_NMS_THRESHOLD        0.3
GPU_COUNT                      1
IMAGES_PER_GPU                 2
IMAGE_MAX_DIM                  1024
IMAGE_MIN_DIM                  800
IMAGE_PADDING                  True
IMAGE_SHAPE                    [1024 1024    3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               100
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           bar
NUM_CLASSES                    2
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
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                1000
TRAIN_ROIS_PER_IMAGE           200
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```

Notebook Preferences

In [3]:

```
def get_ax(rows=1, cols=1, size=8):
    """Return a Matplotlib Axes array to be used in
    all visualizations in the notebook. Provide a
    central point to control graph sizes.

    Change the default size attribute to control the size
    of rendered images
    """
    _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
    return ax
```

Dataset

Create a synthetic dataset

Extend the Dataset class and add a method to load the shapes dataset, `load_shapes()`, and override the following methods:

- load_image()
- load_mask()
- image_reference()

In [4]:

```
class BarrierDataset(utils.Dataset):
    def organize_data(self, data_dir, gt_info_file, height, width):
        """Prepare the dataset supporting info which can be useful later.
        height, width: the size of the generated images.
        """

        tList = open(gt_info_file).readlines()
        fName, fLabels = zip(*[(x.split()[0], np.asarray(x.split()[1:], dtype=np
.float16)) for x in tList])
        fLabels = np.asarray(fLabels) # labels
        numSamp = len(fName)
        print('Number of samples: ' + str(numSamp))

        # Add classes
        self.add_class("bar", 1, "barrier")

        # Add images
        # Generate specifications of images. Images are generated on the fly in
 load_image().
        for i in range(numSamp):
            bbx = np.asarray(fLabels[i].reshape((4,2)), dtype=np.int32)
            path = fName[i] # relative path - '/'.join(fName[0].split('/')[-3:])
            self.add_image("bar", image_id=i, path=path, bboxInfo=bbx, width=wid
th, height=height,)

    def load_image(self, image_id):
        """Load the image associated to the prepared data.
        """
        info = self.image_info[image_id]
        file_path = info['path']
        image = cv2.cvtColor(cv2.imread(file_path), cv2.COLOR_BGR2RGB)
```

```
        ht = info['height']
        wd = info['width']

        image = cv2.resize(image, (ht, wd))

        return image

    def image_reference(self, image_id):
        """Return the shapes data of the image."""
        info = self.image_info[image_id]
        if info["source"] == "bar":
            return info['path']
        else:
            super(self.__class__).image_reference(self, image_id)

    def load_mask(self, image_id):
        """Generate instance masks for shapes of the given image ID.
        """
        info = self.image_info[image_id]
        tbbx = info['bboxInfo']
        path = info['path']
        ht = info['height']
        wd = info['width']

        image = cv2.imread(path)
        mask = np.zeros([ht, wd, 1], dtype=np.uint8)

        # Correct coordinates
        scFact_ht = ht/float(image.shape[0])
        scFact_wd = wd/float(image.shape[1])

        tbbx[:,0] = tbbx[:,0] * scFact_wd
        tbbx[:,1] = tbbx[:,1] * scFact_ht

        tpts = tbbx.reshape((-1,1,2))
        cv2.fillConvexPoly(mask, tpts, (1))

        class_ids = np.array([1])
        return mask, class_ids.astype(np.int32)
```

In [5]:

```
# Prepare datasets
data_dir = '/cyclope/Hasnat/BDDB'
imgHt = 224
imgWd = 224
# Training dataset
gt_info_file_train = '/cyclope/Hasnat/BarDet/tr_test_data/list_BD_20_02_sm_tr.tx
t'
dataset_train = BarrierDataset()
dataset_train.organize_data(data_dir, gt_info_file_train, imgHt, imgWd)
dataset_train.prepare()

# Test dataset
gt_info_file_test = '/cyclope/Hasnat/BarDet/tr_test_data/list_BD_20_02_sm_tst.tx
t'
dataset_val = BarrierDataset()
dataset_val.organize_data(data_dir, gt_info_file_test, imgHt, imgWd)
dataset_val.prepare()
```

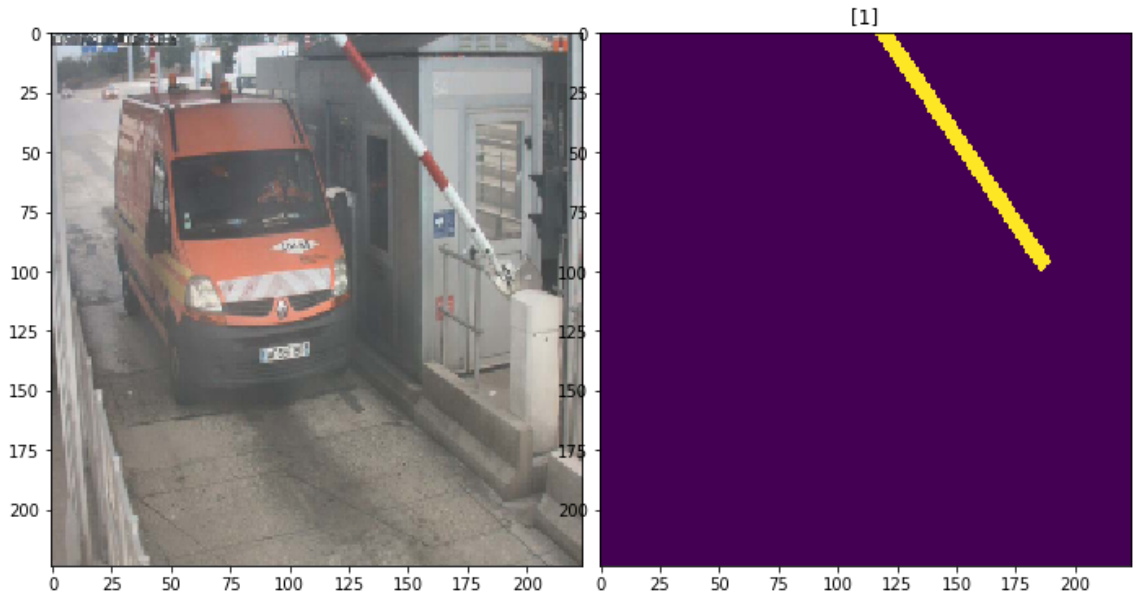
```
Number of samples: 2242
Number of samples: 400
```

In [6]:

```
#### Visualize correctness of data

ii = np.random.choice(dataset_train.image_ids, 1)[0]
tImg = dataset_train.load_image(ii)
tMask, tId = dataset_train.load_mask(ii)

# cv2.rectangle(tImg, (min(tbbx[:,0]), min(tbbx[:,1])), (max(tbbx[:,0]), max(tbb
x[:,1])), (0, 255, 0), 1)

#Show the image and corresponding masks

fig=plt.figure(figsize=(12, 6))
fig.subplots_adjust(hspace=0.01, wspace=0.01)
fig.add_subplot(1, 2, 1)
plt.imshow(tImg)
fig.add_subplot(1, 2, 2)
plt.imshow(tMask[:,:,0])
plt.title(str(tId))
```

Out[6]:

```
Text(0.5,1,'[1]')
```



Create Model

In [7]:

```
# Create model in training mode
model = modellib.MaskRCNN(mode="training", config=config,
                          model_dir=MODEL_DIR)
```

In [8]:

```
# Which weights to start with?
init_with = "imagenet"  # imagenet, coco, or last

if init_with == "imagenet":
    model.load_weights(model.get_imagenet_weights(), by_name=True)
elif init_with == "coco":
    # Load weights trained on MS COCO, but skip layers that
    # are different due to the different number of classes
    # See README for instructions to download the COCO weights
    model.load_weights(COCO_MODEL_PATH, by_name=True,
                       exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",
                                "mrcnn_bbox", "mrcnn_mask"])
elif init_with == "last":
    # Load the last model you trained and continue training
    model.load_weights(model.find_last()[1], by_name=True)
```

```
Downloading data from https://github.com/fchollet/deep-learning-mode
ls/releases/download/v0.2/resnet50_weights_tf_dim_ordering_tf_kernel
s_notop.h5
94658560/94653016 [==============================] - 6s 0us/step
```

Training

Train in two stages:

1. Only the heads. Here we're freezing all the backbone layers and training only the randomly initialized layers (i.e. the ones that we didn't use pre-trained weights from MS COCO). To train only the head layers, pass `layers='heads'` to the `train()` function.
2. Fine-tune all layers. For this simple example it's not necessary, but we're including it to show the process. Simply pass `layers="all` to train all layers.

In [ ]:

```
# Train the head branches
# Passing layers="heads" freezes all layers except the head
# layers. You can also pass a regular expression to select
# which layers to train by name pattern.
model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE,
            epochs=1,
            layers='heads')
```

```
Starting at epoch 0. LR=0.001

Checkpoint Path: /cyclope/Hasnat/Mask_RCNN/logs/bar20180301T1004/mas
k_rcnn_bar_{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)

/usr/local/lib/python3.5/dist-packages/tensorflow/python/ops/gradien
ts_impl.py:96: UserWarning: Converting sparse IndexedSlices to a den
se Tensor of unknown shape. This may consume a large amount of memor
y.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shap
e. "
/usr/local/lib/python3.5/dist-packages/keras/engine/training.py:209
5: UserWarning: Using a generator with `use_multiprocessing=True` an
d multiple workers may duplicate your data. Please consider using th
e`keras.utils.Sequence class.
  UserWarning('Using a generator with `use_multiprocessing=True`'

Epoch 1/1
```

In [ ]:

```
# Fine tune all layers
# Passing layers="all" trains all layers. You can also
# pass a regular expression to select which layers to
# train by name pattern.
model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE / 10,
            epochs=2,
            layers="all")
```

In [ ]:

```
# Save weights
# Typically not needed because callbacks save after every epoch
# Uncomment to save manually
# model_path = os.path.join(MODEL_DIR, "mask_rcnn_shapes.h5")
# model.keras_model.save_weights(model_path)
```

Detection

In [ ]:

```
class InferenceConfig(ShapesConfig):
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

inference_config = InferenceConfig()

# Recreate the model in inference mode
model = modellib.MaskRCNN(mode="inference",
                          config=inference_config,
                          model_dir=MODEL_DIR)

# Get path to saved weights
# Either set a specific path or find last trained weights
# model_path = os.path.join(ROOT_DIR, ".h5 file name here")
model_path = model.find_last()[1]

# Load trained weights (fill in path to trained weights here)
assert model_path != "", "Provide path to trained weights"
print("Loading weights from ", model_path)
model.load_weights(model_path, by_name=True)
```