train_shapes - ACTIVE

April 24, 2018

1 Mask R-CNN - Train on Shapes Dataset

1.0.1 Notes from implementation

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]:
```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))

%matplotlib inline
%load_ext autoreload
%autoreload 2
import os
import sys
import random
import math
import re
import  gc
import time
import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import pprint
import keras.backend as KB
sys.path.append('../')

import mrcnn.model     as modellib
```

```
import mrcnn.visualize as visualize
import mrcnn.shapes    as shapes
from mrcnn.config      import Config
from mrcnn.model       import log
from mrcnn.dataset     import Dataset

from mrcnn.utils       import stack_tensors, stack_tensors_3d
from mrcnn.datagen     import data_generator, load_image_gt
from mrcnn.callbacks   import get_layer_output_1,get_layer_output_2
from mrcnn.visualize   import plot_gaussian
# from mrcnn.pc_layer    import PCTensor
# from mrcnn.pc_layer   import PCNLayer

# Root directory of the project
ROOT_DIR = os.getcwd()
MODEL_PATH = 'E:\Models'
# Directory to save logs and trained model
MODEL_DIR = os.path.join(MODEL_PATH, "mrcnn_logs")
# Path to COCO trained weights
COCO_MODEL_PATH   = os.path.join(MODEL_PATH, "mask_rcnn_coco.h5")
RESNET_MODEL_PATH = os.path.join(MODEL_PATH, "resnet50_weights_tf_dim_ordering_tf_kern

print("Tensorflow Version: {}   Keras Version : {} ".format(tf.__version__,keras.__vers
pp = pprint.PrettyPrinter(indent=2, width=100)
np.set_printoptions(linewidth=100,precision=4)


# Build configuration object -------------------------------------------------
config = shapes.ShapesConfig()
config.BATCH_SIZE      = 5                  # Batch size is 2 (# GPUs * images/GPU).
config.IMAGES_PER_GPU  = 5                  # Must match BATCH_SIZE
config.STEPS_PER_EPOCH = 2
config.FCN_INPUT_SHAPE = config.IMAGE_SHAPE[0:2]
config.display()

# Build shape dataset        -----------------------------------------------
# Training dataset
# generate 500 shapes
dataset_train = shapes.ShapesDataset()
dataset_train.load_shapes(500, config.IMAGE_SHAPE[0], config.IMAGE_SHAPE[1])
dataset_train.prepare()

# Validation dataset
dataset_val = shapes.ShapesDataset()
dataset_val.load_shapes(50, config.IMAGE_SHAPE[0], config.IMAGE_SHAPE[1])
dataset_val.prepare()

try :
```

```
            del model, train_generator, val_generator, mm
            gc.collect()
        except:
            pass
        # Load and display random samples
        # image_ids = np.random.choice(dataset_train.image_ids, 3)
        # for image_id in [3]:
        #     image = dataset_train.load_image(image_id)
        #     mask, class_ids = dataset_train.load_mask(image_id)
        #     visualize.display_top_masks(image, mask, class_ids, dataset_train.class_names)
        print(' COCO Model Path       : ', COCO_MODEL_PATH)
        print(' Checkpoint folder Path: ', MODEL_DIR)
```

```
<IPython.core.display.HTML object>
```

```
D:\Program Files\Anaconda3\envs\TF_gpu\lib\site-packages\h5py\__init__.py:36: FutureWarning: Co
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
Tensorflow Version: 1.6.0   Keras Version : 2.1.3

Configurations:
BACKBONE_SHAPES                [[32 32]
 [16 16]
 [ 8  8]
 [ 4  4]
 [ 2  2]]
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     5
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
FCN_INPUT_SHAPE                [128 128]
GPU_COUNT                      1
IMAGES_PER_GPU                 5
IMAGE_MAX_DIM                  128
IMAGE_MIN_DIM                  128
IMAGE_PADDING                  True
IMAGE_SHAPE                    [128 128   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                           shapes
NUM_CLASSES                    4
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              (8, 16, 32, 64, 128)
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                2
TRAIN_ROIS_PER_IMAGE           32
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               5
WEIGHT_DECAY                   0.0001
```

```
COCO Model Path       :  E:\Models\mask_rcnn_coco.h5
Checkpoint folder Path:  E:\Models\mrcnn_logs
```

2 Create Model

In [2]:
```
try :
    del model
    gc.collect()
except:
    pass

model = modellib.MaskRCNN(mode="training", config=config, model_dir=MODEL_DIR)
#model.keras_model.summary(line_length = 120)

# Which weights to start with?
init_with = "last"  # imagenet, coco, or last
if init_with == "coco":
    # Load weights trained on MS COCO, but skip layers that are different due to the d
    # See README for instructions to download the COCO weights
    loc=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
    loc= model.load_weights(model.find_last()[1], by_name=True)
```

```
        model.compile_only(learning_rate=config.LEARNING_RATE, layers='heads')
        KB.set_learning_phase(1)

>>> Set_log_dir() -- model dir is  E:\Models\mrcnn_logs
    set_log_dir: Checkpoint path set to : E:\Models\mrcnn_logs\shapes20180424T1033\mask_rcnn_sh

>>> Resnet Graph
     Input_image shape : (?, 128, 128, 3)
     After ZeroPadding2D  : (?, 134, 134, 3) (?, 134, 134, 3)
     After Conv2D padding : (?, 64, 64, 64) (?, 64, 64, 64)
     After BatchNorm      : (?, 64, 64, 64) (?, 64, 64, 64)
     After MaxPooling2D   : (?, 32, 32, 64) (?, 32, 32, 64)

>>> Generate pyramid anchors
      Anchor  scales:   (8, 16, 32, 64, 128)
      Anchor  ratios:   [0.5, 1, 2]
      Anchor  stride:   1
      Feature shapes:   [[32 32]
 [16 16]
 [ 8  8]
 [ 4  4]
 [ 2  2]]
     Feature strides:  [4, 8, 16, 32, 64]
    Size of anchor array is : (4092, 4)

>>> RPN Layer
     Input_feature_map shape : (?, ?, ?, 256)
     anchors_per_location    : 3
     depth                   : 256
     Input_feature_map shape : (?, ?, ?, 256)
     anchors_per_location    : 3
     anchor_stride           : 1

>>> RPN Outputs  <class 'list'>
      rpn_class_logits/concat:0
      rpn_class/concat:0
      rpn_bbox/concat:0

>>> Proposal Layer
    Init complete. Size of anchors:  (4092, 4)
     Scores :  (5, 4092)
     Deltas :  (5, 4092, 4)
     Anchors:  (5, 4092, 4)
     Boxes shape / type after processing:  (5, 4092, 4) <class 'tensorflow.python.framework.ops

>>> Detection Target Layer
>>> Detection Target Layer : call  <class 'list'> 4
```

```
     proposals.shape    : (5, ?, ?) <class 'tensorflow.python.framework.ops.Tensor'>
     gt_class_ids.shape : (?, ?) <class 'tensorflow.python.framework.ops.Tensor'>
     gt_bboxes.shape    : (?, ?, 4) <class 'tensorflow.python.framework.ops.Tensor'>
     gt_masks.shape     : (?, 56, 56, ?) <class 'tensorflow.python.framework.ops.Tensor'>
>>> detection_targets_graph - calculate Overlaps_graph
     overlaps_graph: shape of boxes1 before reshape:  (?, ?)
     overlaps_graph: shape of boxes2 before reshape:  (?, 4)
     overlaps_graph: shape of boxes1 after reshape:  (?, 4)
     overlaps_graph: shape of boxes2 after reshape:  (?, 4)
     Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_5:0", shape=(2,)
>>> detection_targets_graph - calculate Overlaps_graph
     overlaps_graph: shape of boxes1 before reshape:  (?, ?)
     overlaps_graph: shape of boxes2 before reshape:  (?, 4)
     overlaps_graph: shape of boxes1 after reshape:  (?, 4)
     overlaps_graph: shape of boxes2 after reshape:  (?, 4)
     Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_10:0", shape=(2
 shape of positive overlaps is : (?, ?)
     roi.shape             : (?, ?) Tensor("proposal_targets/Shape_15:0", shape=(2,), dtype=int
     roi_gt_class_ids.shape: (?,) Tensor("proposal_targets/Shape_16:0", shape=(1,), dtype=int32
     deltas.shape          : (?, ?) Tensor("proposal_targets/Shape_17:0", shape=(2,), dtype=int
     masks.shape           : (?, ?, ?) Tensor("proposal_targets/Shape_18:0", shape=(3,), dtype=
     roi_gt_boxes.shape    : (?, ?) Tensor("proposal_targets/Shape_19:0", shape=(2,), dtype=int
>>> detection_targets_graph - calculate Overlaps_graph
     overlaps_graph: shape of boxes1 before reshape:  (?, ?)
     overlaps_graph: shape of boxes2 before reshape:  (?, 4)
     overlaps_graph: shape of boxes1 after reshape:  (?, 4)
     overlaps_graph: shape of boxes2 after reshape:  (?, 4)
     Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_25:0", shape=(2
>>> detection_targets_graph - calculate Overlaps_graph
     overlaps_graph: shape of boxes1 before reshape:  (?, ?)
     overlaps_graph: shape of boxes2 before reshape:  (?, 4)
     overlaps_graph: shape of boxes1 after reshape:  (?, 4)
     overlaps_graph: shape of boxes2 after reshape:  (?, 4)
     Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_30:0", shape=(2
 shape of positive overlaps is : (?, ?)
     roi.shape             : (?, ?) Tensor("proposal_targets/Shape_35:0", shape=(2,), dtype=int
     roi_gt_class_ids.shape: (?,) Tensor("proposal_targets/Shape_36:0", shape=(1,), dtype=int32
     deltas.shape          : (?, ?) Tensor("proposal_targets/Shape_37:0", shape=(2,), dtype=int
     masks.shape           : (?, ?, ?) Tensor("proposal_targets/Shape_38:0", shape=(3,), dtype=
     roi_gt_boxes.shape    : (?, ?) Tensor("proposal_targets/Shape_39:0", shape=(2,), dtype=int
>>> detection_targets_graph - calculate Overlaps_graph
     overlaps_graph: shape of boxes1 before reshape:  (?, ?)
     overlaps_graph: shape of boxes2 before reshape:  (?, 4)
     overlaps_graph: shape of boxes1 after reshape:  (?, 4)
     overlaps_graph: shape of boxes2 after reshape:  (?, 4)
     Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_45:0", shape=(2
>>> detection_targets_graph - calculate Overlaps_graph
     overlaps_graph: shape of boxes1 before reshape:  (?, ?)
```

```
    overlaps_graph: shape of boxes2 before reshape:  (?, 4)
    overlaps_graph: shape of boxes1 after reshape:  (?, 4)
    overlaps_graph: shape of boxes2 after reshape:  (?, 4)
    Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_50:0", shape=(2
 shape of positive overlaps is : (?, ?)
    roi.shape             : (?, ?) Tensor("proposal_targets/Shape_55:0", shape=(2,), dtype=int
    roi_gt_class_ids.shape: (?,) Tensor("proposal_targets/Shape_56:0", shape=(1,), dtype=int32
    deltas.shape          : (?, ?) Tensor("proposal_targets/Shape_57:0", shape=(2,), dtype=int
    masks.shape           : (?, ?, ?) Tensor("proposal_targets/Shape_58:0", shape=(3,), dtype=
    roi_gt_boxes.shape    : (?, ?) Tensor("proposal_targets/Shape_59:0", shape=(2,), dtype=int
>>> detection_targets_graph - calculate Overlaps_graph
    overlaps_graph: shape of boxes1 before reshape:  (?, ?)
    overlaps_graph: shape of boxes2 before reshape:  (?, 4)
    overlaps_graph: shape of boxes1 after reshape:  (?, 4)
    overlaps_graph: shape of boxes2 after reshape:  (?, 4)
    Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_65:0", shape=(2
>>> detection_targets_graph - calculate Overlaps_graph
    overlaps_graph: shape of boxes1 before reshape:  (?, ?)
    overlaps_graph: shape of boxes2 before reshape:  (?, 4)
    overlaps_graph: shape of boxes1 after reshape:  (?, 4)
    overlaps_graph: shape of boxes2 after reshape:  (?, 4)
    Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_70:0", shape=(2
 shape of positive overlaps is : (?, ?)
    roi.shape             : (?, ?) Tensor("proposal_targets/Shape_75:0", shape=(2,), dtype=int
    roi_gt_class_ids.shape: (?,) Tensor("proposal_targets/Shape_76:0", shape=(1,), dtype=int32
    deltas.shape          : (?, ?) Tensor("proposal_targets/Shape_77:0", shape=(2,), dtype=int
    masks.shape           : (?, ?, ?) Tensor("proposal_targets/Shape_78:0", shape=(3,), dtype=
    roi_gt_boxes.shape    : (?, ?) Tensor("proposal_targets/Shape_79:0", shape=(2,), dtype=int
>>> detection_targets_graph - calculate Overlaps_graph
    overlaps_graph: shape of boxes1 before reshape:  (?, ?)
    overlaps_graph: shape of boxes2 before reshape:  (?, 4)
    overlaps_graph: shape of boxes1 after reshape:  (?, 4)
    overlaps_graph: shape of boxes2 after reshape:  (?, 4)
    Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_85:0", shape=(2
>>> detection_targets_graph - calculate Overlaps_graph
    overlaps_graph: shape of boxes1 before reshape:  (?, ?)
    overlaps_graph: shape of boxes2 before reshape:  (?, 4)
    overlaps_graph: shape of boxes1 after reshape:  (?, 4)
    overlaps_graph: shape of boxes2 after reshape:  (?, 4)
    Overlaps_graph(): Shape of output overlaps Tensor("proposal_targets/Shape_90:0", shape=(2
 shape of positive overlaps is : (?, ?)
    roi.shape             : (?, ?) Tensor("proposal_targets/Shape_95:0", shape=(2,), dtype=int
    roi_gt_class_ids.shape: (?,) Tensor("proposal_targets/Shape_96:0", shape=(1,), dtype=int32
    deltas.shape          : (?, ?) Tensor("proposal_targets/Shape_97:0", shape=(2,), dtype=int
    masks.shape           : (?, ?, ?) Tensor("proposal_targets/Shape_98:0", shape=(3,), dtype=
    roi_gt_boxes.shape    : (?, ?) Tensor("proposal_targets/Shape_99:0", shape=(2,), dtype=int
>>> Detection Target Layer : return call  <class 'list'> 5
    output 0  shape (5, ?, ?)  type <class 'tensorflow.python.framework.ops.Tensor'>
```

```
     output 1  shape (5, ?)  type <class 'tensorflow.python.framework.ops.Tensor'>
     output 2  shape (5, ?, ?)  type <class 'tensorflow.python.framework.ops.Tensor'>
     output 3  shape (5, ?, ?, ?)  type <class 'tensorflow.python.framework.ops.Tensor'>
     output 4  shape (5, ?, ?)  type <class 'tensorflow.python.framework.ops.Tensor'>

>>> FPN Classifier Graph
     rois shape          : (5, ?, ?)
     feature_maps : 4
     feature_maps shape  : (?, 32, 32, 256)
     feature_maps shape  : (?, 16, 16, 256)
     feature_maps shape  : (?, 8, 8, 256)
     feature_maps shape  : (?, 4, 4, 256)
     input_shape         : [128 128   3]
     pool_size           : 7

>>> PCN Layer
>>> PCN Layer : call  <class 'list'> 5
     mrcnn_class.shape    : (?, 32, 4) <class 'tensorflow.python.framework.ops.Tensor'>
     mrcnn_bbox.shape     : (?, 32, 4, 4) <class 'tensorflow.python.framework.ops.Tensor'>
     output_rois.shape    : (?, ?) <class 'tensorflow.python.framework.ops.Tensor'>

>>> PCN Layer TF
>>> PCN Layer TF: call  <class 'list'> 5
     mrcnn_class.shape    : (?, 32, 4) <class 'tensorflow.python.framework.ops.Tensor'>
     mrcnn_bbox.shape     : (?, 32, 4, 4) <class 'tensorflow.python.framework.ops.Tensor'>
     output_rois.shape    : (5, ?, ?) <class 'tensorflow.python.framework.ops.Tensor'>
     gt_class_ids.shape   : (?, ?) <class 'tensorflow.python.framework.ops.Tensor'>
     gt_bboxes.shape      : (?, ?, 4) <class 'tensorflow.python.framework.ops.Tensor'>

    *** build_predictions_tf
    pred_array shape: (5, 32, 7)
    pred_scatter shape is  (5, 4, 32, 7) Tensor("cntxt_layer_2/ScatterNd:0", shape=(5, 4, 32,
    sort inds shape :  (5, 4, 32)
    class_grid   <class 'tensorflow.python.framework.ops.Tensor'> shape (5, 4, 32)
    batch_grid   <class 'tensorflow.python.framework.ops.Tensor'> shape (5, 4, 32)
    roi_grid shape (5, 4, 32) roi_grid_exp shape  (5, 4, 32, 1)
    gather_inds  <class 'tensorflow.python.framework.ops.Tensor'> shape (5, 4, 32, 3)
    pred_tensor (gathered)  :  (5, 4, 32, 7)
    -- pred_tensor results (bboxes sorted by score) ----
    final pred_tensor shape  :  (5, 4, 32, 8)
    final pred_cls_cnt shape :  (5, 4)
    complete

    *** build_ground_truth_tf
    gt_class_ids shape :  (?, ?)     notm_gt_bbox.shape  :  (?, ?, 4)
    gt_classes_exp shape  (?, ?, 1)
    pred_ scores shape  (?, ?)
    bbox_idx shape     (5, 100, 1)
```

```
gt_array shape     (5, 100, 7)
bbox_grid  shape   (5, 100)
batch_grid shape   (5, 100)
scatter_ind shape  (5, 100, 3)
gt_scatter shape  (5, 4, 100, 7)
build gathering indexes to use in sorting -------
sort inds shape :  (5, 4, 100)
class_grid  shape  (5, 4, 100)
batch_grid  shape  (5, 4, 100)
bbox_grid   shape  (5, 4, 100)  bbox_grid_exp shape  (5, 4, 100, 1)
gather_inds shape  (5, 4, 100, 3)
gt_tensor (gathered)   :  (5, 4, 100, 8)
final gt_tensor shape  :  (5, 4, 100, 8)
final gt_cls_cnt shape :  (5, 4)
complete

*** build_gaussian_tf
in_tensor shape :  (5, 4, 32, 8)
modified in_tensor shape :  (5, 4, 32, 5)
num of bboxes per class is :  Tensor("cntxt_layer_2/ToInt32_1/x:0", shape=(), dtype=int32)
after transpose  (128, 128, 5, 32, 2)
pt2_sum shape  (5, 4, 32)
dense shape  (?, 6)
Build Stacked output from dynamically partitioned lists --------------

>> input to MVN.PROB: pos_grid (meshgrid) shape:  (128, 128, 5, 32, 2)
<< output probabilities shape: (128, 128, 5, 32)
Scatter out the probability distributions based on class --------------
gaussian_grid      :  (5, 32, 128, 128)
class shape        :  (5, ?)
roi_grid shape     :  (5, 32)
batch_grid shape   :  (5, 32)
scatter_classes    :  (5, 32, 3)
gaussian scattered :  (5, 4, 32, 128, 128)
Reduce sum based on class ------------------------------------------
gaussian_sum shape :  (5, 128, 128, 4)
complete

*** build_gaussian_tf
in_tensor shape :  (5, 4, 100, 8)
modified in_tensor shape :  (5, 4, 100, 5)
num of bboxes per class is :  Tensor("cntxt_layer_2/ToInt32_4/x:0", shape=(), dtype=int32)
after transpose  (128, 128, 5, 100, 2)
pt2_sum shape  (5, 4, 100)
dense shape  (?, 6)
Build Stacked output from dynamically partitioned lists --------------

>> input to MVN.PROB: pos_grid (meshgrid) shape:  (128, 128, 5, 100, 2)
```

```
    << output probabilities shape: (128, 128, 5, 100)
    Scatter out the probability distributions based on class --------------
    gaussian_grid      :  (5, 100, 128, 128)
    class shape        :  (5, ?)
    roi_grid shape     :  (5, 100)
    batch_grid shape   :  (5, 100)
    scatter_classes    :  (5, 100, 3)
    gaussian scattered :  (5, 4, 100, 128, 128)
    Reduce sum based on class ------------------------------------------------
    gaussian_sum shape :  (5, 128, 128, 4)
    complete
   Output build_gaussian_tf (ground truth)
     gt_gaussian :  (5, 128, 128, 4)
<<<  shape of gt_gaussian_2      (5, 128, 128, 4)
<<<  shape of gt_tensor2         (5, 4, 100, 8)
<<<  shape of gt_cls_cnt2        (5, 4)

>>> FCN Layer
     feature map shape is  (5, 128, 128, 4)
     height : 128 width : 128 classes : 4
     image_data_format     channels_last
   FCN Block 11 shape is :  (5, 128, 128, 64)
   FCN Block 12 shape is :  (5, 128, 128, 64)
   FCN Block 13 shape is :  (5, 64, 64, 64)
   FCN Block 21 shape is :  (5, 64, 64, 128)
   FCN Block 22 shape is :  (5, 64, 64, 128)
   FCN Block 23 (Max pooling) shape is :  (5, 32, 32, 128)
   FCN Block 31 shape is :  (5, 32, 32, 256)
   FCN Block 32 shape is :  (5, 32, 32, 256)
   FCN Block 33 shape is :  (5, 32, 32, 256)
   FCN Block 34 (Max pooling) shape is :  (5, 16, 16, 256)
   FCN fully connected 1 (fcn_fc1) shape is :  (5, 16, 16, 2048)
   FCN fully connected 2 (fcn_fc2) shape is :  (5, 16, 16, 2048)
   FCN final conv2d (fcn_classify) shape is :  (None, 16, 16, 4)
   h_factor :  8.0 w_factor :  8.0

>>> BilinearUpSampling2D layer
     data_format :  channels_last
     size        :  (8.0, 8.0)
     target_size :  None
     input_spec  :  [InputSpec(ndim=4)]
     call resize_images_bilinear with size:  (8.0, 8.0)
     CHANNELS LAST: X:  (5, 16, 16, 4)  KB.int_shape() :  (None, 16, 16, 4)
     target_height   :  None  target_width  :  None
     new_shape (2):  (2,) [16 16]
     new_shape (3):  (2,) [128 128]
     X after image.resize_bilinear:  (5, ?, ?, 4)
     Dimensions of X after set_shape() :  (5, 128, 128, 4)
```

```
    BilinearUpSampling2D. compute_output_shape()
   FCN output (fcn_bilinear) shape is :  (5, 128, 128, 4)
>>> rpn_bbox_loss_graph
    rpn_match size : (?, ?)
    rpn_bbox  size : (?, ?, 4)
    tf default session:  <tensorflow.python.client.session.InteractiveSession object at 0x0000
>>> rpn_bbox_loss_graph
    rpn_match size : (?, ?)
    rpn_bbox  size : (?, ?, 4)
    tf default session:  <tensorflow.python.client.session.InteractiveSession object at 0x0000
>>> mrcnn_class_loss_graph
    target_class_ids  size : (5, ?)
    pred_class_logits size : (?, 32, 4)
    active_class_ids  size : (?, ?)
>>> mrcnn_class_loss_graph
    target_class_ids  size : (?, 1)
    pred_class_logits size : (?, 32, 4)
    active_class_ids  size : (?, ?)
>>> mrcnn_bbox_loss_graph
    target_class_ids  size : (5, ?)
    pred_bbox size         : (?, 32, 4, 4)
    target_bbox size       : (5, ?, ?)
>>> mrcnn_bbox_loss_graph
    target_class_ids  size : (?, 1)
    pred_bbox size         : (?, 32, 4, 4)
    target_bbox size       : (?, 32, 4)
>>> mrcnn_mask_loss_graph
    target_class_ids shape : (5, ?)
    target_masks     shape : (5, ?, ?, ?)
    pred_masks       shape : (?, 32, 28, 28, 4)
    target_class_ids shape : (?,)
    mask_shape       shape : (4,)
    target_masks     shape : (?, 32, 28, 28, 4)
    pred_shape       shape : (5,)
    pred_masks       shape : (?, ?, ?, ?)
     y_true shape: (?, ?, ?)
     y_pred shape: (?, ?, ?)
     final loss shape: (1, 1)
>>> mrcnn_mask_loss_graph
    target_class_ids shape : (?, 1)
    target_masks     shape : (?, 32, 28, 28)
    pred_masks       shape : (?, 32, 28, 28, 4)
    target_class_ids shape : (?,)
    mask_shape       shape : (4,)
    target_masks     shape : (?, 32, 28, 28, 4)
    pred_shape       shape : (5,)
    pred_masks       shape : (?, ?, ?, ?)
     y_true shape: (?, ?, ?)
```

```
     y_pred shape: (?, ?, ?)
     final loss shape: (1, 1)
>>> MaskRCNN build complete
>>> MaskRCNN initialization complete
>>> find_last checkpoint file()
    find_last info:   dir_name: E:\Models\mrcnn_logs\shapes20180313T1856
    find_last info: checkpoint: E:\Models\mrcnn_logs\shapes20180313T1856\mask_rcnn_shapes_0242
>>> load_weights()
    load_weights: Loading weights from: E:\Models\mrcnn_logs\shapes20180313T1856\mask_rcnn_shap
    load_weights: Log directory set to : E:\Models\mrcnn_logs\shapes20180313T1856\mask_rcnn_sha
>>> Set_log_dir() -- model dir is  E:\Models\mrcnn_logs
    set_log_dir: model_path (input) is : E:/Models/mrcnn_logs/shapes20180313T1856/mask_rcnn_sha
    set_log_dir: self.epoch set to 243  (Next epoch to run)
    set_log_dir: tensorboard path: E:\Models\mrcnn_logs\tensorboard
    set_log_dir: Checkpoint path set to : E:\Models\mrcnn_logs\shapes20180313T1856\mask_rcnn_sh
>>> Load weights complete
Compile with learing rate; 0.001 Learning Moementum: 0.9
Checkpoint Folder:  E:\Models\mrcnn_logs\shapes20180313T1856\mask_rcnn_shapes_{epoch:04d}.h5

Selecting layers to train
Layer    Layer Name              Layer Type
174  fpn_c5p5              (Conv2D)
176  fpn_c4p4              (Conv2D)
179  fpn_c3p3              (Conv2D)
182  fpn_c2p2              (Conv2D)
184  fpn_p5                (Conv2D)
185  fpn_p2                (Conv2D)
186  fpn_p3                (Conv2D)
187  fpn_p4                (Conv2D)
In model:  rpn_model
      1  rpn_conv_shared        (Conv2D)
      2  rpn_class_raw          (Conv2D)
      4  rpn_bbox_pred          (Conv2D)
199  mrcnn_class_conv1     (TimeDistributed)
200  mrcnn_class_bn1       (TimeDistributed)
202  mrcnn_class_conv2     (TimeDistributed)
203  mrcnn_class_bn2       (TimeDistributed)
206  mrcnn_class_logits    (TimeDistributed)
207  mrcnn_bbox_fc         (TimeDistributed)
214  mrcnn_mask_conv1      (TimeDistributed)
216  mrcnn_mask_bn1        (TimeDistributed)
220  mrcnn_mask_conv2      (TimeDistributed)
222  mrcnn_mask_bn2        (TimeDistributed)
226  mrcnn_mask_conv3      (TimeDistributed)
228  mrcnn_mask_bn3        (TimeDistributed)
232  mrcnn_mask_conv4      (TimeDistributed)
234  mrcnn_mask_bn4        (TimeDistributed)
238  mrcnn_mask_deconv     (TimeDistributed)
```

```
242  mrcnn_mask             (TimeDistributed)
```

2.0.1 Print some model information

```
In [ ]: # print('\n Learning phase values is L ' ,KB.learning_phase())
        # print('\n Metrics (_get_deduped_metrics_names():) ')
        # pp.pprint(model.keras_model._get_deduped_metrics_names())
        # print('\n Outputs: ')
        # pp.pprint(model.keras_model.outputs)
        # print('\n Losses (model.metrics_names): ')
        # pp.pprint(model.keras_model.metrics_names)

        model.keras_model.summary(line_length = 150)
```

2.0.2 Define Data Generator

```
In [3]: train_generator = data_generator(dataset_train, model.config, shuffle=True,
                                         batch_size=model.config.BATCH_SIZE,
                                         augment = False)
        val_generator = data_generator(dataset_val, model.config, shuffle=True,
                                       batch_size=model.config.BATCH_SIZE,
                                       augment=False)
```

2.0.3 Get next shapes from generator and display loaded shapes

```
In [4]: train_batch_x, train_batch_y = next(train_generator)
```

```
>>> Generate pyramid anchors
      Anchor  scales:   (8, 16, 32, 64, 128)
      Anchor  ratios:   [0.5, 1, 2]
      Anchor  stride:   1
      Feature shapes:   [[32 32]
 [16 16]
 [ 8  8]
 [ 4  4]
 [ 2  2]]
      Feature strides:  [4, 8, 16, 32, 64]
    Size of anchor array is : (4092, 4)
```

```
In [5]: # train_batch_x, train_batch_y = next(train_generator)
        imgmeta_idx = model.keras_model.input_names.index('input_image_meta')
        img_meta    = train_batch_x[imgmeta_idx]

        for img_idx in range(config.BATCH_SIZE):
            image_id = img_meta[img_idx,0]
            image = dataset_train.load_image(image_id)
```