Standard Model -

Using TensorFlow and Mask R-CNN to detect and classify astronomical objects from Hubble Space Telescope data

In [1]:

```
# Import required modules
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
import tensorflow as tf

# Root directory of the project
ROOT_DIR = os.path.abspath("/gpfs01/home/ppzsb1/astobjdet/Mask_RCNN")

# Import Mask RCNN
sys.path.append(ROOT_DIR)  # To find local version of the library
from mrcnn.config import Config
from mrcnn import utils
import mrcnn.model as modellib
from mrcnn import visualize
from mrcnn.model import log

%matplotlib inline

# 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)




# if multiple GPUs, only use one of them. Select 0 or 1 for either GPU.
os.environ["CUDA_VISIBLE_DEVICES"]="1"

# avoid hogging all the GPU memory
#config = tf.ConfigProto()
#config.gpu_options.allow_growth=True
#sess = tf.Session(config=config)
```

```
Using TensorFlow backend.
```

In [2]:

```
# check that we have the devices we expect available
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

Out[2]:

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 7499031979713504542, name: "/device:XLA_CPU:0"
 device_type: "XLA_CPU"
 memory_limit: 17179869184
 locality {
```

```
 locality {
 }
 incarnation: 6190468239821169937
 physical_device_desc: "device: XLA_CPU device", name: "/device:XLA_GPU:0"
 device_type: "XLA_GPU"
 memory_limit: 17179869184
 locality {
 }
 incarnation: 10069691959182558367
 physical_device_desc: "device: XLA_GPU device", name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 15596303156
 locality {
   bus_id: 2
   numa_node: 1
   links {
   }
 }
 incarnation: 2017778128315991308
 physical_device_desc: "device: 0, name: Tesla V100-PCIE-16GB, pci bus id: 0000:d8:00.0, compute c
apability: 7.0"]
```

In [3]:

```python
from astropy.io import fits
from astropy.visualization.lupton_rgb import make_lupton_rgb
from astropy.table import Table
from matplotlib.image import imsave
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
from glob import glob
import random
import os.path

def crops(fnames, xrange, yrange):
    return [fits.open(f)[0].data[xrange[0]:xrange[1], yrange[0]:yrange[1]] for f in fnames]

class AstroObjectDataset(utils.Dataset):

    def load_hst_images(self, classtype='object',
                        cropsize=256, datadir='/gpfs01/home/ppzsb1/astobjdet/images/',
                        fieldglob='*', cropglob='crops_standard_m0_s0.5_q8',
                        training=True, train_frac=0.8):
        datatable = Table.read(os.path.join(datadir,'mastersex.fits'))
        datatable.convert_bytestring_to_unicode()
        # Add classes
        if classtype == 'object':
            self.add_class("astroobject", 1, "object")
            self.nclass = 1
            datatable['object_class'] = 1
            classcolumn = 'object_class'
        elif classtype == 'mag':
            self.add_class("astroobject", 1, "faint")
            self.add_class("astroobject", 2, "medium")
            self.add_class("astroobject", 3, "bright")
            self.nclass = 3
            classcolumn = 'mag_class'
        elif classtype == 'colour':
            self.add_class("astroobject", 1, "blue")
            self.add_class("astroobject", 2, "red")
            self.nclass = 2
            classcolumn = 'colour_class'
        elif classtype == 'conc':
            self.add_class("astroobject", 1, "lowconc")
            self.add_class("astroobject", 2, "highconc")
            self.nclass = 2
            classcolumn = 'conc_class'
        elif classtype == 'colourconcmag':
            # example of combining to create a new set of classes
            datatable['colourconcmag_class'] = 0
            c = 0
            for i, colour in enumerate(['blue', 'red']):
                for j, conc in enumerate(['lowconc', 'highconc']):
                    for k, mag in enumerate(['faint', 'medium', 'bright']):
                        c += 1
                        select = ((datatable['colour_class'] == i+1) &
                        select = ((datatable['colour_class'] == i+1) &
                                  (datatable['conc_class'] == j+1) &
                                  (datatable['mag_class'] == k+1))
                        datatable['colourconcmag_class'][select] = c
                        self.add_class("astroobject", c, "{}_{}_{}".format(colour, conc, mag))
            self.nclass = c
            classcolumn = 'colourconcmag_class'

        datatable = datatable.to_pandas().set_index(['field', 'NUMBER'])
        self.objclass = datatable[classcolumn]


        # Add images
        imageglob = os.path.join(datadir, '{}/{}/crop*npy'.format(fieldglob, cropglob))
        images = glob(imageglob)
        random.seed(12345)
        random.shuffle(images)
        print('Total number of images:', len(images))
        if training:
            start = 0
            stop = int(train_frac * len(images))
        else:
            start = int(train_frac * len(images))
            stop = len(images)
        count = 0
        for k in range(start, stop):
            count += 1
            path = images[k]
            field = path.split('/')[-3]
            self.add_image("astroobject", image_id=k, path=path,
                           field=field,
                           width=cropsize, height=cropsize)
        print('Number of images used:', count)

    def load_image(self, image_id):
        info = self.image_info[image_id]
        img = np.load(info['path'])
        return img

    def load_mask(self, image_id):
        info = self.image_info[image_id]
        seg = np.load(info['path'].replace('/crop_', '/mask_'))
        segids = list(set(seg.ravel()) - set([0]))
        nobj = len(segids)
        mask = np.zeros(seg.shape + (nobj,), np.bool)
        class_ids = np.zeros(nobj, dtype=np.int32)
        for i in range(nobj):
            mask[:,:,i] = seg == segids[i]
            idi = self.objclass.loc[info['field'], segids[i]]
            class_ids[i] = idi
        return mask, class_ids
```

In [4]:

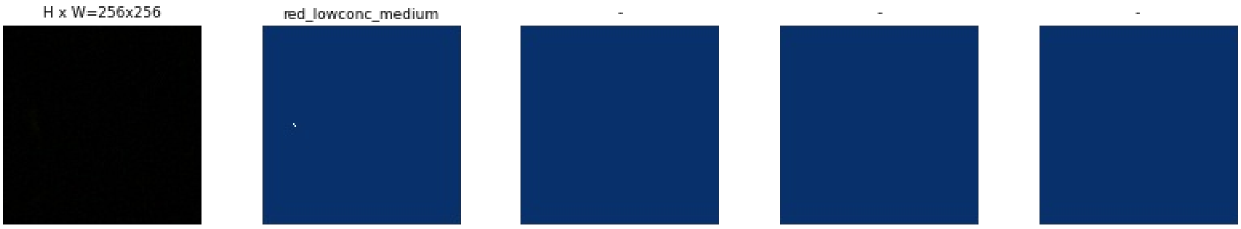
```
classtype = 'colourconcmag'

# Training dataset
dataset_train = AstroObjectDataset()
dataset_train.load_hst_images(classtype, training=True)
dataset_train.prepare()

# Validation dataset
dataset_val = AstroObjectDataset()
dataset_val.load_hst_images(classtype, training=False)
dataset_val.prepare()
```

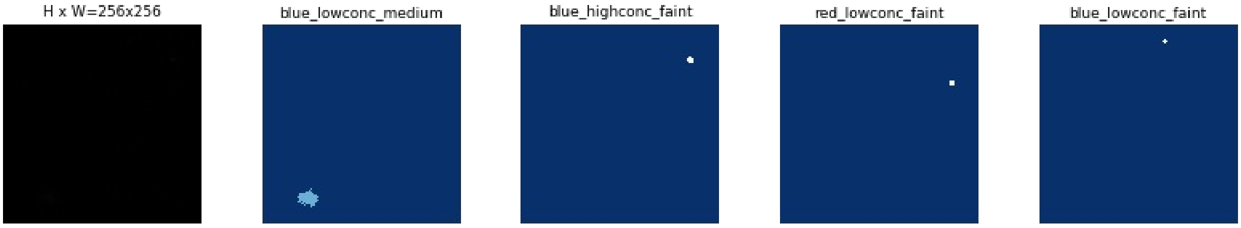
```
Total number of images: 9834
Number of images used: 7867
Total number of images: 9834
Number of images used: 1967
```

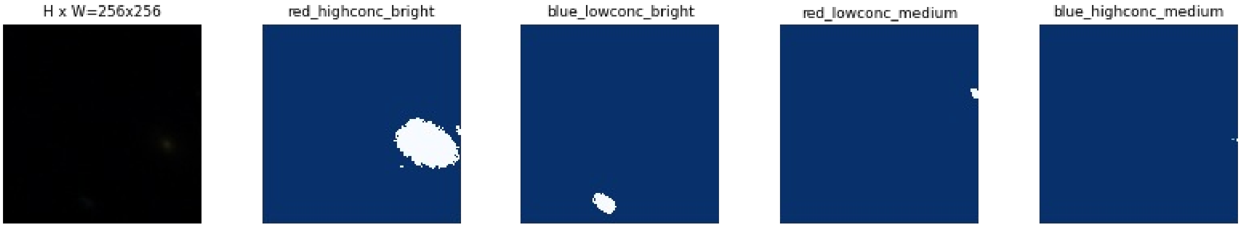
In [5]:

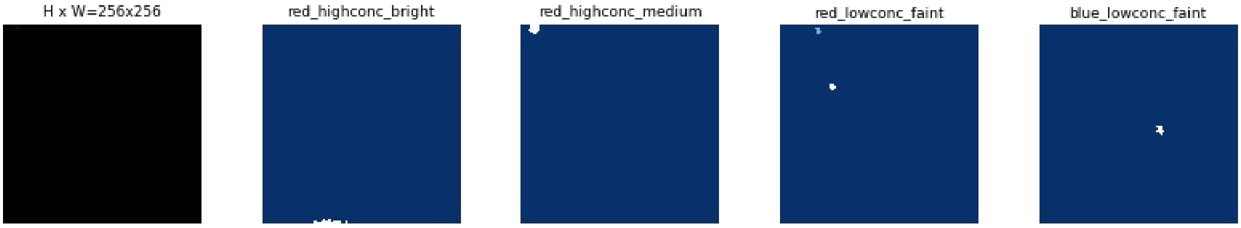
```
# Load and display random samples
image_ids = np.random.choice(dataset_train.image_ids, 4)
```

```
for image_id in image_ids:
    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)
```









Configurations

In [6]:

```
class AstroObjectsConfig(Config):
    """Configuration for training on the astroobjects dataset.
    Derives from the base Config class and overrides values specific
    to the dataset.
    """
    # Give the configuration a recognizable name
    NAME = "crops_standard_m0_s0.5_q8"

    # Backbone network architecture
    # Supported values are: resnet50, resnet101
    BACKBONE = "resnet101"

    # 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).
    GPU_COUNT = 1
    IMAGES_PER_GPU = 4

    # Number of classes (including background)
    NUM_CLASSES = 1 + dataset_train.nclass  # background + 1 object

    # Use small images for faster training. Set the limits of the small side
    # the large side, and that determines the image shape.
    IMAGE_MIN_DIM = 256
    IMAGE_MAX_DIM = 256
    IMAGE_MAX_DIM = 256

    # Use smaller anchors because our image and objects are small
    RPN_ANCHOR_SCALES = (8, 16, 32, 64, 128)  # anchor side in pixels

    # Reduce training ROIs per image because the images are small and have
    # few objects. Aim to allow ROI sampling to pick 33% positive ROIs.
    TRAIN_ROIS_PER_IMAGE = 64

    # Use a small epoch since the data is simple
    STEPS_PER_EPOCH = 300

    # use small validation steps since the epoch is small
    VALIDATION_STEPS = 50

config = AstroObjectsConfig()
config.display()
```

```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     4
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.7
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 4
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  256
IMAGE_META_SIZE                25
IMAGE_MIN_DIM                  256
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [256 256   3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
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               100
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           crops_standard_m0_s0.5_q8
NUM_CLASSES                    13
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              (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                300
TOP_DOWN_PYRAMID_SIZE          256
TRAIN_BN                       False
TRAIN_ROIS_PER_IMAGE           64
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```

Helper Functions

In [7]:

```
def get_history(model):
    eventfiles = sorted(glob(model.log_dir + '/events*'))
    history = {}
    for f in eventfiles:
        for e in tf.train.summary_iterator(f):
            for v in e.summary.value:
                if v.tag in history:
                    history[v.tag].append(v.simple_value)
                else:
                    history[v.tag] = [v.simple_value]
    return history
```

In [8]:

```
def histplot(history):
    hist = pd.DataFrame(history)
    n = len(hist.columns)//2
    fig, axarr = plt.subplots(n, 1, sharex=True,
                                   figsize=(10, 5*n))
    k = 0
    for c in hist.columns:
        if 'val' not in c:
            hist.plot(y=[c, 'val_'+c], ax=axarr[k])
            axarr[k].set_xlabel('epoch')
            axarr[k].set_ylabel(c)
            axarr[k].legend(loc='upper right')
            k += 1
    plt.tight_layout()
```

Create Model

In [9]:

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

In [10]:

```
# Which weights to start with?
init_with = "coco"  # 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(), by_name=True)
```

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 [11]:

```
# 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=50,
            layers='heads')
```

```
Starting at epoch 0. LR=0.001

Checkpoint Path:
/gpfs01/home/ppzsb1/astobjdet/Mask_RCNN/logs/crops_standard_m0_s0.5_q8_standardmodel20190509T1933/m
rcnn_crops_standard_m0_s0.5_q8_standardmodel_{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)
```

```
/gpfs01/home/ppzsb1/.conda/envs/astobjdet/lib/python3.6/site-
packages/tensorflow/python/ops/gradients_impl.py:112: UserWarning: Converting sparse IndexedSlices
to a dense Tensor of unknown shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
/gpfs01/home/ppzsb1/.conda/envs/astobjdet/lib/python3.6/site-
packages/keras/engine/training_generator.py:47: UserWarning: Using a generator with
`use_multiprocessing=True` and multiple workers may duplicate your data. Please consider using the
`keras.utils.Sequence class.
  UserWarning('Using a generator with `use_multiprocessing=True`'
```

```
Epoch 1/50
300/300 [==============================] - 117s 391ms/step - loss: 2.2608 - rpn_class_loss: 0.1334
- rpn_bbox_loss: 0.8650 - mrcnn_class_loss: 0.3678 - mrcnn_bbox_loss: 0.3802 - mrcnn_mask_loss: 0.
5144 - val_loss: 3.1603 - val_rpn_class_loss: 0.1207 - val_rpn_bbox_loss: 1.0964 -
val_mrcnn_class_loss: 0.5672 - val_mrcnn_bbox_loss: 0.8360 - val_mrcnn_mask_loss: 0.5401
Epoch 2/50
300/300 [==============================] - 84s 281ms/step - loss: 1.7667 - rpn_class_loss: 0.0845
- rpn_bbox_loss: 0.6696 - mrcnn_class_loss: 0.3002 - mrcnn_bbox_loss: 0.2935 - mrcnn_mask_loss: 0.
4189 - val_loss: 3.1530 - val_rpn_class_loss: 0.1032 - val_rpn_bbox_loss: 1.0405 -
val_mrcnn_class_loss: 0.7193 - val_mrcnn_bbox_loss: 0.6950 - val_mrcnn_mask_loss: 0.5950
Epoch 3/50
300/300 [==============================] - 84s 279ms/step - loss: 1.7812 - rpn_class_loss: 0.0836
- rpn_bbox_loss: 0.6538 - mrcnn_class_loss: 0.3759 - mrcnn_bbox_loss: 0.2764 - mrcnn_mask_loss: 0.
3915 - val_loss: 3.4937 - val_rpn_class_loss: 0.1023 - val_rpn_bbox_loss: 1.4518 -
val_mrcnn_class_loss: 0.6035 - val_mrcnn_bbox_loss: 0.7385 - val_mrcnn_mask_loss: 0.5976
Epoch 4/50
300/300 [==============================] - 84s 280ms/step - loss: 1.9991 - rpn_class_loss: 0.0829
- rpn_bbox_loss: 0.8778 - mrcnn_class_loss: 0.3986 - mrcnn_bbox_loss: 0.2815 - mrcnn_mask_loss: 0.
3583 - val_loss: 3.4005 - val_rpn_class_loss: 0.0954 - val_rpn_bbox_loss: 1.2411 -
val_mrcnn_class_loss: 0.9814 - val_mrcnn_bbox_loss: 0.5990 - val_mrcnn_mask_loss: 0.4836
Epoch 5/50
300/300 [==============================] - 84s 279ms/step - loss: 1.7345 - rpn_class_loss: 0.0613
```

```
300/300 [==============================] - 84s 279ms/step - loss: 1.7545 - rpn_class_loss: 0.0615
- rpn_bbox_loss: 0.6268 - mrcnn_class_loss: 0.3747 - mrcnn_bbox_loss: 0.2907 - mrcnn_mask_loss: 0.
3810 - val_loss: 2.7851 - val_rpn_class_loss: 0.0630 - val_rpn_bbox_loss: 1.0175 -
val_mrcnn_class_loss: 0.5882 - val_mrcnn_bbox_loss: 0.5974 - val_mrcnn_mask_loss: 0.5189
Epoch 6/50
300/300 [==============================] - 84s 279ms/step - loss: 1.7485 - rpn_class_loss: 0.0738
- rpn_bbox_loss: 0.6419 - mrcnn_class_loss: 0.3890 - mrcnn_bbox_loss: 0.2680 - mrcnn_mask_loss: 0.
3758 - val_loss: 3.3238 - val_rpn_class_loss: 0.0851 - val_rpn_bbox_loss: 1.2115 -
val_mrcnn_class_loss: 0.8887 - val_mrcnn_bbox_loss: 0.6322 - val_mrcnn_mask_loss: 0.5063
Epoch 7/50
300/300 [==============================] - 84s 280ms/step - loss: 1.4820 - rpn_class_loss: 0.0547
- rpn_bbox_loss: 0.5182 - mrcnn_class_loss: 0.3474 - mrcnn_bbox_loss: 0.2241 - mrcnn_mask_loss: 0.
3376 - val_loss: 3.4022 - val_rpn_class_loss: 0.0732 - val_rpn_bbox_loss: 1.1469 -
val_mrcnn_class_loss: 1.1201 - val_mrcnn_bbox_loss: 0.6028 - val_mrcnn_mask_loss: 0.4592
Epoch 8/50
300/300 [==============================] - 84s 279ms/step - loss: 1.5492 - rpn_class_loss: 0.0629
- rpn_bbox_loss: 0.5608 - mrcnn_class_loss: 0.3645 - mrcnn_bbox_loss: 0.2262 - mrcnn_mask_loss: 0.
3348 - val_loss: 3.0746 - val_rpn_class_loss: 0.0539 - val_rpn_bbox_loss: 0.9213 -
val_mrcnn_class_loss: 0.7654 - val_mrcnn_bbox_loss: 0.7537 - val_mrcnn_mask_loss: 0.5803
Epoch 9/50
300/300 [==============================] - 84s 280ms/step - loss: 1.5495 - rpn_class_loss: 0.0608
- rpn_bbox_loss: 0.4760 - mrcnn_class_loss: 0.4114 - mrcnn_bbox_loss: 0.2491 - mrcnn_mask_loss: 0.
3522 - val_loss: 2.8181 - val_rpn_class_loss: 0.0716 - val_rpn_bbox_loss: 0.9440 -
val_mrcnn_class_loss: 0.8541 - val_mrcnn_bbox_loss: 0.5446 - val_mrcnn_mask_loss: 0.4038
Epoch 10/50
300/300 [==============================] - 85s 282ms/step - loss: 1.6147 - rpn_class_loss: 0.0694
- rpn_bbox_loss: 0.5835 - mrcnn_class_loss: 0.3501 - mrcnn_bbox_loss: 0.2462 - mrcnn_mask_loss: 0.
3655 - val_loss: 3.2355 - val_rpn_class_loss: 0.0567 - val_rpn_bbox_loss: 1.2308 -
val_mrcnn_class_loss: 0.8225 - val_mrcnn_bbox_loss: 0.6634 - val_mrcnn_mask_loss: 0.4620
Epoch 11/50
300/300 [==============================] - 84s 280ms/step - loss: 1.5264 - rpn_class_loss: 0.0523
- rpn_bbox_loss: 0.5326 - mrcnn_class_loss: 0.3480 - mrcnn_bbox_loss: 0.2370 - mrcnn_mask_loss: 0.
3565 - val_loss: 2.9513 - val_rpn_class_loss: 0.0535 - val_rpn_bbox_loss: 1.0707 -
val_mrcnn_class_loss: 0.6323 - val_mrcnn_bbox_loss: 0.7225 - val_mrcnn_mask_loss: 0.4723
Epoch 12/50
300/300 [==============================] - 84s 280ms/step - loss: 1.5694 - rpn_class_loss: 0.0598
- rpn_bbox_loss: 0.5894 - mrcnn_class_loss: 0.3280 - mrcnn_bbox_loss: 0.2384 - mrcnn_mask_loss: 0.
3538 - val_loss: 3.2720 - val_rpn_class_loss: 0.0543 - val_rpn_bbox_loss: 1.1445 -
val_mrcnn_class_loss: 1.0449 - val_mrcnn_bbox_loss: 0.5656 - val_mrcnn_mask_loss: 0.4627
Epoch 13/50
300/300 [==============================] - 84s 280ms/step - loss: 1.4280 - rpn_class_loss: 0.0527
- rpn_bbox_loss: 0.4845 - mrcnn_class_loss: 0.3278 - mrcnn_bbox_loss: 0.2280 - mrcnn_mask_loss: 0.
3350 - val_loss: 3.1077 - val_rpn_class_loss: 0.0805 - val_rpn_bbox_loss: 1.1705 -
val_mrcnn_class_loss: 0.7943 - val_mrcnn_bbox_loss: 0.5688 - val_mrcnn_mask_loss: 0.4937
Epoch 14/50
300/300 [==============================] - 84s 281ms/step - loss: 1.2750 - rpn_class_loss: 0.0409
- rpn_bbox_loss: 0.4184 - mrcnn_class_loss: 0.2722 - mrcnn_bbox_loss: 0.2095 - mrcnn_mask_loss: 0.
3340 - val_loss: 3.3151 - val_rpn_class_loss: 0.0499 - val_rpn_bbox_loss: 0.9211 -
val_mrcnn_class_loss: 1.1528 - val_mrcnn_bbox_loss: 0.5895 - val_mrcnn_mask_loss: 0.6019
Epoch 15/50
300/300 [==============================] - 84s 279ms/step - loss: 1.4928 - rpn_class_loss: 0.0510
- rpn_bbox_loss: 0.4567 - mrcnn_class_loss: 0.3955 - mrcnn_bbox_loss: 0.2472 - mrcnn_mask_loss: 0.
3423 - val_loss: 3.3977 - val_rpn_class_loss: 0.0847 - val_rpn_bbox_loss: 1.1728 -
val_mrcnn_class_loss: 0.8604 - val_mrcnn_bbox_loss: 0.7541 - val_mrcnn_mask_loss: 0.5257
Epoch 16/50
300/300 [==============================] - 84s 281ms/step - loss: 1.4217 - rpn_class_loss: 0.0486
- rpn_bbox_loss: 0.4478 - mrcnn_class_loss: 0.3482 - mrcnn_bbox_loss: 0.2387 - mrcnn_mask_loss: 0.
3384 - val_loss: 3.7460 - val_rpn_class_loss: 0.1417 - val_rpn_bbox_loss: 1.3888 -
val_mrcnn_class_loss: 1.1289 - val_mrcnn_bbox_loss: 0.6083 - val_mrcnn_mask_loss: 0.4783
Epoch 17/50
300/300 [==============================] - 84s 279ms/step - loss: 1.4197 - rpn_class_loss: 0.0485
- rpn_bbox_loss: 0.4964 - mrcnn_class_loss: 0.3350 - mrcnn_bbox_loss: 0.2147 - mrcnn_mask_loss: 0.
3251 - val_loss: 3.5501 - val_rpn_class_loss: 0.0671 - val_rpn_bbox_loss: 1.2235 -
val_mrcnn_class_loss: 1.1407 - val_mrcnn_bbox_loss: 0.5926 - val_mrcnn_mask_loss: 0.5262
Epoch 18/50
300/300 [==============================] - 85s 282ms/step - loss: 1.4237 - rpn_class_loss: 0.0440
- rpn_bbox_loss: 0.5416 - mrcnn_class_loss: 0.2941 - mrcnn_bbox_loss: 0.2154 - mrcnn_mask_loss: 0.
3285 - val_loss: 3.1731 - val_rpn_class_loss: 0.0603 - val_rpn_bbox_loss: 1.1258 -
val_mrcnn_class_loss: 0.8212 - val_mrcnn_bbox_loss: 0.6800 - val_mrcnn_mask_loss: 0.4858
Epoch 19/50
300/300 [==============================] - 84s 281ms/step - loss: 1.3296 - rpn_class_loss: 0.0450
- rpn_bbox_loss: 0.4239 - mrcnn_class_loss: 0.2951 - mrcnn_bbox_loss: 0.2279 - mrcnn_mask_loss: 0.
3378 - val_loss: 2.9485 - val_rpn_class_loss: 0.0610 - val_rpn_bbox_loss: 1.0935 -
val_mrcnn_class_loss: 0.6501 - val_mrcnn_bbox_loss: 0.6443 - val_mrcnn_mask_loss: 0.4995
Epoch 20/50
300/300 [==============================] - 84s 280ms/step - loss: 1.2932 - rpn_class_loss: 0.0478
- rpn_bbox_loss: 0.4192 - mrcnn_class_loss: 0.3127 - mrcnn_bbox_loss: 0.2144 - mrcnn_mask_loss: 0.
2991 - val_loss: 3.5364 - val_rpn_class_loss: 0.0677 - val_rpn_bbox_loss: 1.6799 -
```

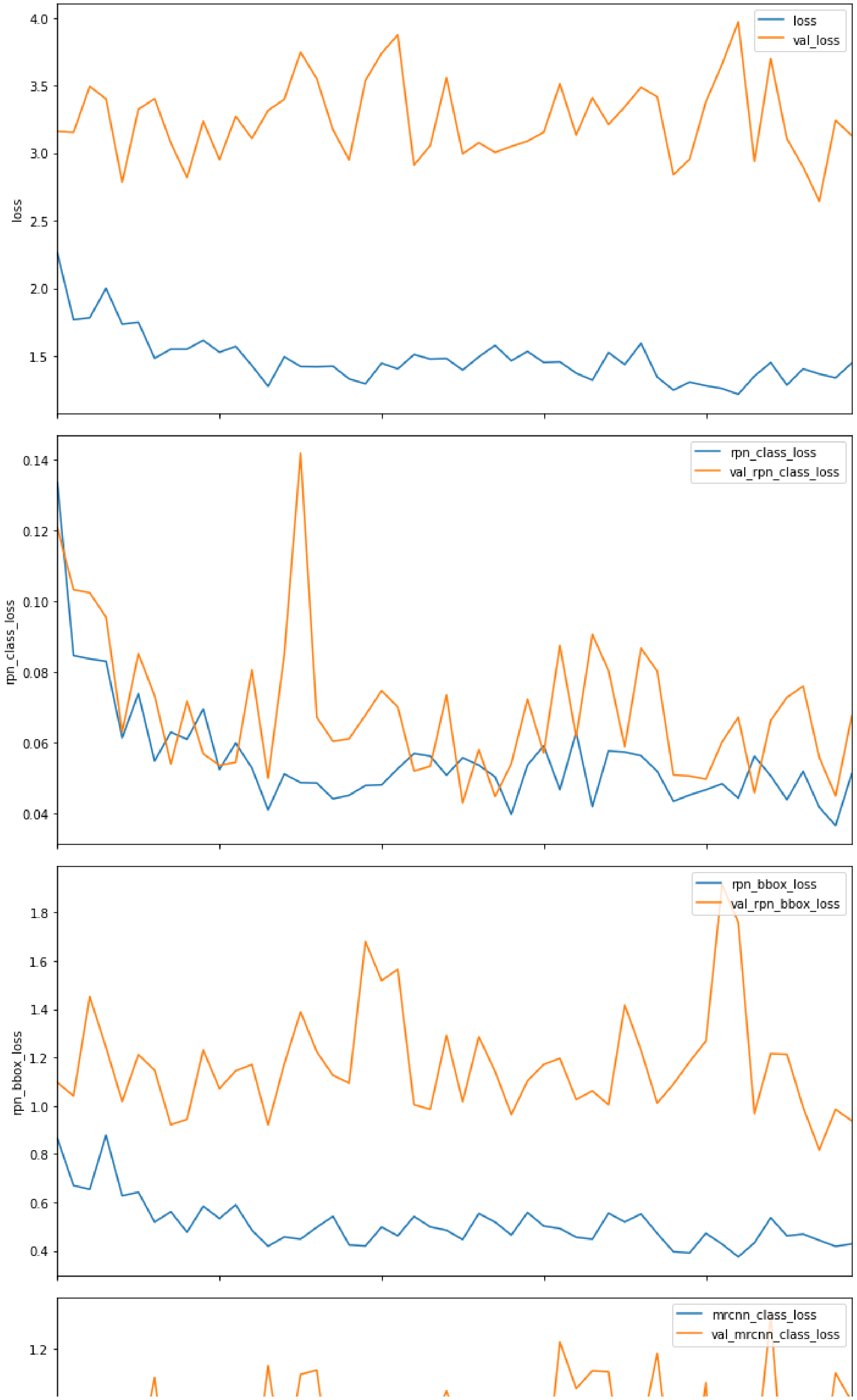
```
2991 - val_loss: 3.5564 - val_rpn_class_loss: 0.0677 - val_rpn_bbox_loss: 1.6799 -
val_mrcnn_class_loss: 0.8334 - val_mrcnn_bbox_loss: 0.5268 - val_mrcnn_mask_loss: 0.4286
Epoch 21/50
300/300 [==============================] - 84s 280ms/step - loss: 1.4455 - rpn_class_loss: 0.0479
- rpn_bbox_loss: 0.4975 - mrcnn_class_loss: 0.3298 - mrcnn_bbox_loss: 0.2390 - mrcnn_mask_loss: 0.
3312 - val_loss: 3.7381 - val_rpn_class_loss: 0.0746 - val_rpn_bbox_loss: 1.5180 -
val_mrcnn_class_loss: 0.8204 - val_mrcnn_bbox_loss: 0.7384 - val_mrcnn_mask_loss: 0.5867
Epoch 22/50
300/300 [==============================] - 85s 283ms/step - loss: 1.4037 - rpn_class_loss: 0.0525
- rpn_bbox_loss: 0.4610 - mrcnn_class_loss: 0.3086 - mrcnn_bbox_loss: 0.2299 - mrcnn_mask_loss: 0.
3515 - val_loss: 3.8747 - val_rpn_class_loss: 0.0700 - val_rpn_bbox_loss: 1.5653 -
val_mrcnn_class_loss: 0.9423 - val_mrcnn_bbox_loss: 0.6338 - val_mrcnn_mask_loss: 0.6634
Epoch 23/50
300/300 [==============================] - 84s 279ms/step - loss: 1.5100 - rpn_class_loss: 0.0568
- rpn_bbox_loss: 0.5416 - mrcnn_class_loss: 0.3436 - mrcnn_bbox_loss: 0.2310 - mrcnn_mask_loss: 0.
3370 - val_loss: 2.9106 - val_rpn_class_loss: 0.0519 - val_rpn_bbox_loss: 1.0042 -
val_mrcnn_class_loss: 0.6689 - val_mrcnn_bbox_loss: 0.5859 - val_mrcnn_mask_loss: 0.5998
Epoch 24/50
300/300 [==============================] - 84s 281ms/step - loss: 1.4764 - rpn_class_loss: 0.0561
- rpn_bbox_loss: 0.4982 - mrcnn_class_loss: 0.3417 - mrcnn_bbox_loss: 0.2352 - mrcnn_mask_loss: 0.
3451 - val_loss: 3.0551 - val_rpn_class_loss: 0.0532 - val_rpn_bbox_loss: 0.9853 -
val_mrcnn_class_loss: 0.9769 - val_mrcnn_bbox_loss: 0.5994 - val_mrcnn_mask_loss: 0.4403
Epoch 25/50
300/300 [==============================] - 84s 281ms/step - loss: 1.4801 - rpn_class_loss: 0.0507
- rpn_bbox_loss: 0.4841 - mrcnn_class_loss: 0.3457 - mrcnn_bbox_loss: 0.2573 - mrcnn_mask_loss: 0.
3423 - val_loss: 3.5567 - val_rpn_class_loss: 0.0734 - val_rpn_bbox_loss: 1.2907 -
val_mrcnn_class_loss: 1.0831 - val_mrcnn_bbox_loss: 0.6435 - val_mrcnn_mask_loss: 0.4659
Epoch 26/50
300/300 [==============================] - 84s 280ms/step - loss: 1.3950 - rpn_class_loss: 0.0556
- rpn_bbox_loss: 0.4459 - mrcnn_class_loss: 0.3431 - mrcnn_bbox_loss: 0.2225 - mrcnn_mask_loss: 0.
3278 - val_loss: 2.9945 - val_rpn_class_loss: 0.0428 - val_rpn_bbox_loss: 1.0165 -
val_mrcnn_class_loss: 0.9530 - val_mrcnn_bbox_loss: 0.5642 - val_mrcnn_mask_loss: 0.4180
Epoch 27/50
300/300 [==============================] - 84s 279ms/step - loss: 1.4928 - rpn_class_loss: 0.0535
- rpn_bbox_loss: 0.5535 - mrcnn_class_loss: 0.3398 - mrcnn_bbox_loss: 0.2300 - mrcnn_mask_loss: 0.
3159 - val_loss: 3.0764 - val_rpn_class_loss: 0.0579 - val_rpn_bbox_loss: 1.2853 -
val_mrcnn_class_loss: 0.7296 - val_mrcnn_bbox_loss: 0.6042 - val_mrcnn_mask_loss: 0.3994
Epoch 28/50
300/300 [==============================] - 84s 281ms/step - loss: 1.5768 - rpn_class_loss: 0.0502
- rpn_bbox_loss: 0.5181 - mrcnn_class_loss: 0.3751 - mrcnn_bbox_loss: 0.2774 - mrcnn_mask_loss: 0.
3560 - val_loss: 3.0050 - val_rpn_class_loss: 0.0447 - val_rpn_bbox_loss: 1.1422 -
val_mrcnn_class_loss: 0.8207 - val_mrcnn_bbox_loss: 0.5509 - val_mrcnn_mask_loss: 0.4464
Epoch 29/50
300/300 [==============================] - 83s 278ms/step - loss: 1.4640 - rpn_class_loss: 0.0396
- rpn_bbox_loss: 0.4642 - mrcnn_class_loss: 0.3581 - mrcnn_bbox_loss: 0.2492 - mrcnn_mask_loss: 0.
3530 - val_loss: 3.0491 - val_rpn_class_loss: 0.0539 - val_rpn_bbox_loss: 0.9637 -
val_mrcnn_class_loss: 1.0451 - val_mrcnn_bbox_loss: 0.5005 - val_mrcnn_mask_loss: 0.4858
Epoch 30/50
300/300 [==============================] - 85s 282ms/step - loss: 1.5323 - rpn_class_loss: 0.0535
- rpn_bbox_loss: 0.5576 - mrcnn_class_loss: 0.3468 - mrcnn_bbox_loss: 0.2433 - mrcnn_mask_loss: 0.
3311 - val_loss: 3.0880 - val_rpn_class_loss: 0.0722 - val_rpn_bbox_loss: 1.1027 -
val_mrcnn_class_loss: 0.9341 - val_mrcnn_bbox_loss: 0.5471 - val_mrcnn_mask_loss: 0.4319
Epoch 31/50
300/300 [==============================] - 84s 281ms/step - loss: 1.4516 - rpn_class_loss: 0.0590
- rpn_bbox_loss: 0.5017 - mrcnn_class_loss: 0.3401 - mrcnn_bbox_loss: 0.2230 - mrcnn_mask_loss: 0.
3278 - val_loss: 3.1537 - val_rpn_class_loss: 0.0570 - val_rpn_bbox_loss: 1.1706 -
val_mrcnn_class_loss: 0.7868 - val_mrcnn_bbox_loss: 0.6564 - val_mrcnn_mask_loss: 0.4828
Epoch 32/50
300/300 [==============================] - 85s 282ms/step - loss: 1.4557 - rpn_class_loss: 0.0466
- rpn_bbox_loss: 0.4914 - mrcnn_class_loss: 0.3436 - mrcnn_bbox_loss: 0.2381 - mrcnn_mask_loss: 0.
3360 - val_loss: 3.5119 - val_rpn_class_loss: 0.0874 - val_rpn_bbox_loss: 1.1961 -
val_mrcnn_class_loss: 1.2184 - val_mrcnn_bbox_loss: 0.5435 - val_mrcnn_mask_loss: 0.4665
Epoch 33/50
300/300 [==============================] - 84s 279ms/step - loss: 1.3720 - rpn_class_loss: 0.0629
- rpn_bbox_loss: 0.4555 - mrcnn_class_loss: 0.2933 - mrcnn_bbox_loss: 0.2275 - mrcnn_mask_loss: 0.
3328 - val_loss: 3.1335 - val_rpn_class_loss: 0.0615 - val_rpn_bbox_loss: 1.0255 -
val_mrcnn_class_loss: 1.0898 - val_mrcnn_bbox_loss: 0.5413 - val_mrcnn_mask_loss: 0.4155
Epoch 34/50
300/300 [==============================] - 84s 280ms/step - loss: 1.3204 - rpn_class_loss: 0.0418
- rpn_bbox_loss: 0.4476 - mrcnn_class_loss: 0.2868 - mrcnn_bbox_loss: 0.2208 - mrcnn_mask_loss: 0.
3234 - val_loss: 3.4098 - val_rpn_class_loss: 0.0905 - val_rpn_bbox_loss: 1.0616 -
val_mrcnn_class_loss: 1.1388 - val_mrcnn_bbox_loss: 0.5495 - val_mrcnn_mask_loss: 0.5694
Epoch 35/50
300/300 [==============================] - 85s 282ms/step - loss: 1.5241 - rpn_class_loss: 0.0575
- rpn_bbox_loss: 0.5547 - mrcnn_class_loss: 0.3469 - mrcnn_bbox_loss: 0.2350 - mrcnn_mask_loss: 0.
3299 - val_loss: 3.2117 - val_rpn_class_loss: 0.0803 - val_rpn_bbox_loss: 1.0037 -
val_mrcnn_class_loss: 1.1361 - val_mrcnn_bbox_loss: 0.5844 - val_mrcnn_mask_loss: 0.4072
Epoch 36/50
Epoch 36/50
300/300 [==============================] - 84s 280ms/step - loss: 1.4353 - rpn_class_loss: 0.0572
- rpn_bbox_loss: 0.5187 - mrcnn_class_loss: 0.3100 - mrcnn_bbox_loss: 0.2270 - mrcnn_mask_loss: 0.
3225 - val_loss: 3.3420 - val_rpn_class_loss: 0.0587 - val_rpn_bbox_loss: 1.4170 -
val_mrcnn_class_loss: 0.8131 - val_mrcnn_bbox_loss: 0.5606 - val_mrcnn_mask_loss: 0.4926
Epoch 37/50
300/300 [==============================] - 84s 279ms/step - loss: 1.5930 - rpn_class_loss: 0.0562
- rpn_bbox_loss: 0.5519 - mrcnn_class_loss: 0.3721 - mrcnn_bbox_loss: 0.2444 - mrcnn_mask_loss: 0.
3684 - val_loss: 3.4868 - val_rpn_class_loss: 0.0866 - val_rpn_bbox_loss: 1.2292 -
val_mrcnn_class_loss: 0.9651 - val_mrcnn_bbox_loss: 0.6969 - val_mrcnn_mask_loss: 0.5090
Epoch 38/50
300/300 [==============================] - 84s 280ms/step - loss: 1.3431 - rpn_class_loss: 0.0518
- rpn_bbox_loss: 0.4713 - mrcnn_class_loss: 0.2983 - mrcnn_bbox_loss: 0.2129 - mrcnn_mask_loss: 0.
3088 - val_loss: 3.4165 - val_rpn_class_loss: 0.0801 - val_rpn_bbox_loss: 1.0110 -
val_mrcnn_class_loss: 1.1864 - val_mrcnn_bbox_loss: 0.6532 - val_mrcnn_mask_loss: 0.4858
Epoch 39/50
300/300 [==============================] - 84s 281ms/step - loss: 1.2467 - rpn_class_loss: 0.0433
- rpn_bbox_loss: 0.3954 - mrcnn_class_loss: 0.2718 - mrcnn_bbox_loss: 0.2104 - mrcnn_mask_loss: 0.
3258 - val_loss: 2.8388 - val_rpn_class_loss: 0.0508 - val_rpn_bbox_loss: 1.0894 -
val_mrcnn_class_loss: 0.8108 - val_mrcnn_bbox_loss: 0.5131 - val_mrcnn_mask_loss: 0.3748
Epoch 40/50
300/300 [==============================] - 84s 279ms/step - loss: 1.3059 - rpn_class_loss: 0.0451
- rpn_bbox_loss: 0.3904 - mrcnn_class_loss: 0.3473 - mrcnn_bbox_loss: 0.2098 - mrcnn_mask_loss: 0.
3133 - val_loss: 2.9544 - val_rpn_class_loss: 0.0505 - val_rpn_bbox_loss: 1.1818 -
val_mrcnn_class_loss: 0.7749 - val_mrcnn_bbox_loss: 0.5416 - val_mrcnn_mask_loss: 0.4057
Epoch 41/50
300/300 [==============================] - 84s 281ms/step - loss: 1.2788 - rpn_class_loss: 0.0466
- rpn_bbox_loss: 0.4719 - mrcnn_class_loss: 0.2614 - mrcnn_bbox_loss: 0.1955 - mrcnn_mask_loss: 0.
3034 - val_loss: 3.3794 - val_rpn_class_loss: 0.0496 - val_rpn_bbox_loss: 1.2679 -
val_mrcnn_class_loss: 1.1061 - val_mrcnn_bbox_loss: 0.5247 - val_mrcnn_mask_loss: 0.4311
Epoch 42/50
300/300 [==============================] - 83s 278ms/step - loss: 1.2584 - rpn_class_loss: 0.0483
- rpn_bbox_loss: 0.4272 - mrcnn_class_loss: 0.2896 - mrcnn_bbox_loss: 0.1921 - mrcnn_mask_loss: 0.
3012 - val_loss: 3.6569 - val_rpn_class_loss: 0.0601 - val_rpn_bbox_loss: 1.9159 -
val_mrcnn_class_loss: 0.4341 - val_mrcnn_bbox_loss: 0.7080 - val_mrcnn_mask_loss: 0.5387
Epoch 43/50
300/300 [==============================] - 84s 281ms/step - loss: 1.2155 - rpn_class_loss: 0.0442
- rpn_bbox_loss: 0.3739 - mrcnn_class_loss: 0.2824 - mrcnn_bbox_loss: 0.2067 - mrcnn_mask_loss: 0.
3083 - val_loss: 3.9700 - val_rpn_class_loss: 0.0671 - val_rpn_bbox_loss: 1.7589 -
val_mrcnn_class_loss: 0.9517 - val_mrcnn_bbox_loss: 0.7355 - val_mrcnn_mask_loss: 0.4568
Epoch 44/50
300/300 [==============================] - 84s 281ms/step - loss: 1.3504 - rpn_class_loss: 0.0561
- rpn_bbox_loss: 0.4327 - mrcnn_class_loss: 0.3042 - mrcnn_bbox_loss: 0.2244 - mrcnn_mask_loss: 0.
3331 - val_loss: 2.9384 - val_rpn_class_loss: 0.0457 - val_rpn_bbox_loss: 0.9677 -
val_mrcnn_class_loss: 0.8662 - val_mrcnn_bbox_loss: 0.5794 - val_mrcnn_mask_loss: 0.4794
Epoch 45/50
300/300 [==============================] - 84s 280ms/step - loss: 1.4505 - rpn_class_loss: 0.0505
- rpn_bbox_loss: 0.5356 - mrcnn_class_loss: 0.3101 - mrcnn_bbox_loss: 0.2289 - mrcnn_mask_loss: 0.
3255 - val_loss: 3.6971 - val_rpn_class_loss: 0.0662 - val_rpn_bbox_loss: 1.2154 -
val_mrcnn_class_loss: 1.2908 - val_mrcnn_bbox_loss: 0.6197 - val_mrcnn_mask_loss: 0.5049
Epoch 46/50
300/300 [==============================] - 83s 278ms/step - loss: 1.2839 - rpn_class_loss: 0.0437
- rpn_bbox_loss: 0.4612 - mrcnn_class_loss: 0.2740 - mrcnn_bbox_loss: 0.1932 - mrcnn_mask_loss: 0.
3118 - val_loss: 3.1060 - val_rpn_class_loss: 0.0727 - val_rpn_bbox_loss: 1.2127 -
val_mrcnn_class_loss: 0.7211 - val_mrcnn_bbox_loss: 0.6516 - val_mrcnn_mask_loss: 0.4479
Epoch 47/50
300/300 [==============================] - 85s 282ms/step - loss: 1.4040 - rpn_class_loss: 0.0518
- rpn_bbox_loss: 0.4684 - mrcnn_class_loss: 0.3304 - mrcnn_bbox_loss: 0.2199 - mrcnn_mask_loss: 0.
3336 - val_loss: 2.8969 - val_rpn_class_loss: 0.0759 - val_rpn_bbox_loss: 0.9938 -
val_mrcnn_class_loss: 0.7931 - val_mrcnn_bbox_loss: 0.6234 - val_mrcnn_mask_loss: 0.4106
Epoch 48/50
300/300 [==============================] - 83s 278ms/step - loss: 1.3663 - rpn_class_loss: 0.0416
- rpn_bbox_loss: 0.4427 - mrcnn_class_loss: 0.3253 - mrcnn_bbox_loss: 0.2167 - mrcnn_mask_loss: 0.
3400 - val_loss: 2.6438 - val_rpn_class_loss: 0.0557 - val_rpn_bbox_loss: 0.8174 -
val_mrcnn_class_loss: 0.8207 - val_mrcnn_bbox_loss: 0.5698 - val_mrcnn_mask_loss: 0.3802
Epoch 49/50
300/300 [==============================] - 84s 280ms/step - loss: 1.3372 - rpn_class_loss: 0.0364
- rpn_bbox_loss: 0.4178 - mrcnn_class_loss: 0.3152 - mrcnn_bbox_loss: 0.2327 - mrcnn_mask_loss: 0.
3351 - val_loss: 3.2412 - val_rpn_class_loss: 0.0448 - val_rpn_bbox_loss: 0.9849 -
val_mrcnn_class_loss: 1.1323 - val_mrcnn_bbox_loss: 0.6580 - val_mrcnn_mask_loss: 0.4211
Epoch 50/50
300/300 [==============================] - 84s 279ms/step - loss: 1.4468 - rpn_class_loss: 0.0512
- rpn_bbox_loss: 0.4280 - mrcnn_class_loss: 0.3689 - mrcnn_bbox_loss: 0.2502 - mrcnn_mask_loss: 0.
3485 - val_loss: 3.1295 - val_rpn_class_loss: 0.0675 - val_rpn_bbox_loss: 0.9375 -
val_mrcnn_class_loss: 1.0533 - val_mrcnn_bbox_loss: 0.5880 - val_mrcnn_mask_loss: 0.4832
```

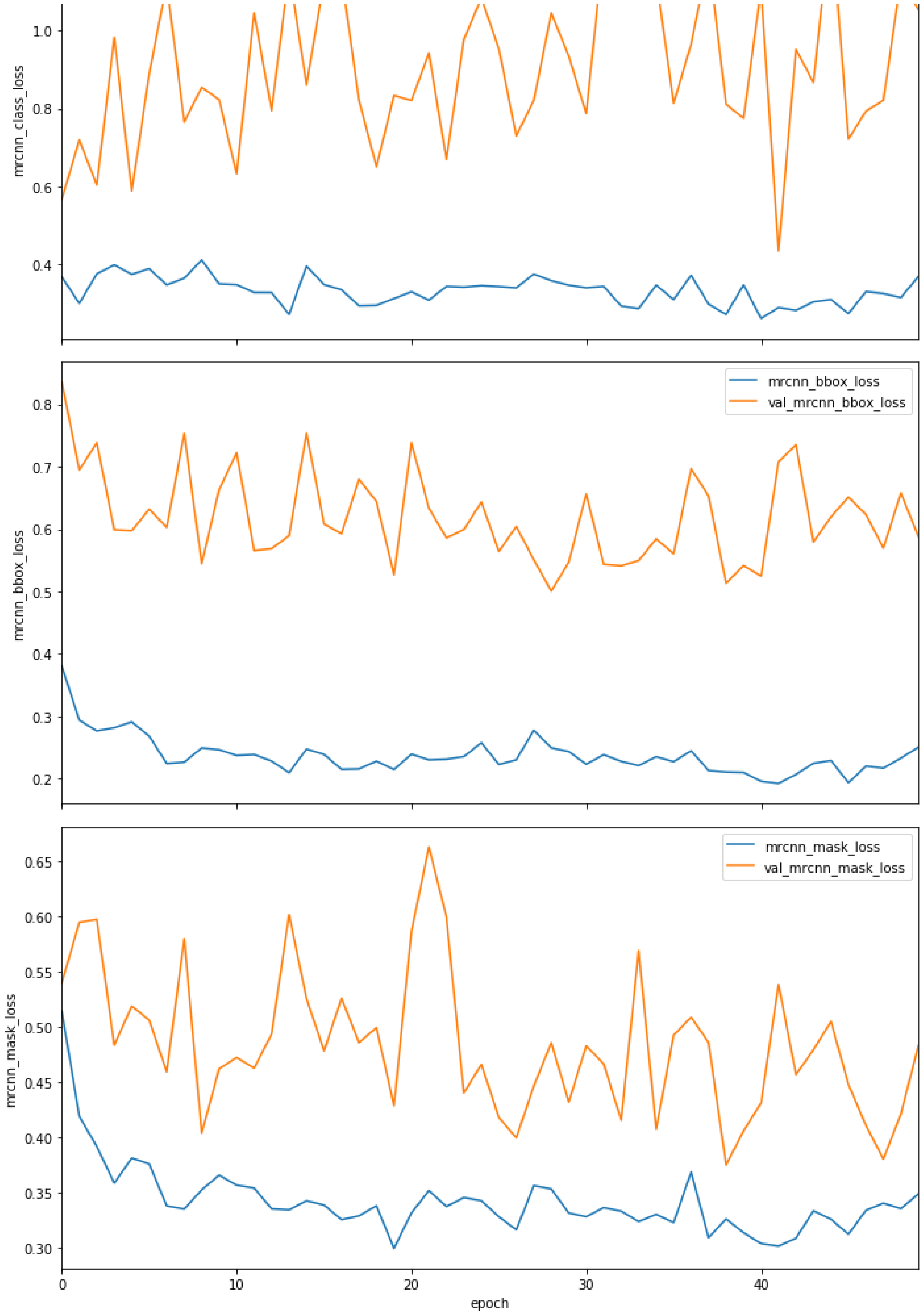
In [12]:

```
history_heads = get_history(model)
```

In [13]:

```
histplot(history_heads)
```





In [14]:

```
# 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=100,
            layers="all")
```

```
Starting at epoch 50. LR=0.0001

Checkpoint Path:
/gpfs01/home/ppzsb1/astobjdet/Mask_RCNN/logs/crops_standard_m0_s0.5_q8_standardmodel20190509T1933/m
rcnn_crops_standard_m0_s0.5_q8_standardmodel_{epoch:04d}.h5
Selecting layers to train
Selecting layers to train
conv1                  (Conv2D)
bn_conv1               (BatchNorm)
res2a_branch2a         (Conv2D)
bn2a_branch2a          (BatchNorm)
res2a_branch2b         (Conv2D)
bn2a_branch2b          (BatchNorm)
res2a_branch2c         (Conv2D)
res2a_branch1          (Conv2D)
bn2a_branch2c          (BatchNorm)
bn2a_branch1           (BatchNorm)
res2b_branch2a         (Conv2D)
bn2b_branch2a          (BatchNorm)
res2b_branch2b         (Conv2D)
bn2b_branch2b          (BatchNorm)
res2b_branch2c         (Conv2D)
bn2b_branch2c          (BatchNorm)
res2c_branch2a         (Conv2D)
bn2c_branch2a          (BatchNorm)
res2c_branch2b         (Conv2D)
bn2c_branch2b          (BatchNorm)
res2c_branch2c         (Conv2D)
bn2c_branch2c          (BatchNorm)
res3a_branch2a         (Conv2D)
bn3a_branch2a          (BatchNorm)
res3a_branch2b         (Conv2D)
bn3a_branch2b          (BatchNorm)
res3a_branch2c         (Conv2D)
res3a_branch1          (Conv2D)
bn3a_branch2c          (BatchNorm)
bn3a_branch1           (BatchNorm)
res3b_branch2a         (Conv2D)
bn3b_branch2a          (BatchNorm)
res3b_branch2b         (Conv2D)
bn3b_branch2b          (BatchNorm)
res3b_branch2c         (Conv2D)
bn3b_branch2c          (BatchNorm)
res3c_branch2a         (Conv2D)
bn3c_branch2a          (BatchNorm)
res3c_branch2b         (Conv2D)
bn3c_branch2b          (BatchNorm)
res3c_branch2c         (Conv2D)
bn3c_branch2c          (BatchNorm)
res3d_branch2a         (Conv2D)
bn3d_branch2a          (BatchNorm)
res3d_branch2b         (Conv2D)
bn3d_branch2b          (BatchNorm)
res3d_branch2c         (Conv2D)
bn3d_branch2c          (BatchNorm)
res4a_branch2a         (Conv2D)
bn4a_branch2a          (BatchNorm)
res4a_branch2b         (Conv2D)
bn4a_branch2b          (BatchNorm)
res4a_branch2c         (Conv2D)
res4a_branch1          (Conv2D)
bn4a_branch2c          (BatchNorm)
bn4a_branch1           (BatchNorm)
res4b_branch2a         (Conv2D)
bn4b_branch2a          (BatchNorm)
res4b_branch2b         (Conv2D)
bn4b_branch2b          (BatchNorm)
res4b_branch2c         (Conv2D)
bn4b_branch2c          (BatchNorm)
res4c_branch2a         (Conv2D)
bn4c_branch2a          (BatchNorm)
res4c_branch2b         (Conv2D)
bn4c_branch2b          (BatchNorm)
res4c_branch2c         (Conv2D)
bn4c_branch2c          (BatchNorm)
res4d_branch2a         (Conv2D)
bn4d_branch2a          (BatchNorm)
res4d_branch2b         (Conv2D)
bn4d_branch2b          (BatchNorm)
res4d_branch2c         (Conv2D)
bn4d_branch2c          (BatchNorm)
res4e_branch2a         (Conv2D)
bn4e_branch2a          (BatchNorm)
res4e_branch2b         (Conv2D)
```

```
res4e_branch2b         (Conv2D)
bn4e_branch2b          (BatchNorm)
res4e_branch2c         (Conv2D)
bn4e_branch2c          (BatchNorm)
res4f_branch2a         (Conv2D)
bn4f_branch2a          (BatchNorm)
res4f_branch2b         (Conv2D)
bn4f_branch2b          (BatchNorm)
res4f_branch2c         (Conv2D)
bn4f_branch2c          (BatchNorm)
res4g_branch2a         (Conv2D)
bn4g_branch2a          (BatchNorm)
res4g_branch2b         (Conv2D)
bn4g_branch2b          (BatchNorm)
res4g_branch2c         (Conv2D)
bn4g_branch2c          (BatchNorm)
res4h_branch2a         (Conv2D)
bn4h_branch2a          (BatchNorm)
res4h_branch2b         (Conv2D)
bn4h_branch2b          (BatchNorm)
res4h_branch2c         (Conv2D)
bn4h_branch2c          (BatchNorm)
res4i_branch2a         (Conv2D)
bn4i_branch2a          (BatchNorm)
res4i_branch2b         (Conv2D)
bn4i_branch2b          (BatchNorm)
res4i_branch2c         (Conv2D)
bn4i_branch2c          (BatchNorm)
res4j_branch2a         (Conv2D)
bn4j_branch2a          (BatchNorm)
res4j_branch2b         (Conv2D)
bn4j_branch2b          (BatchNorm)
res4j_branch2c         (Conv2D)
bn4j_branch2c          (BatchNorm)
res4k_branch2a         (Conv2D)
bn4k_branch2a          (BatchNorm)
res4k_branch2b         (Conv2D)
bn4k_branch2b          (BatchNorm)
res4k_branch2c         (Conv2D)
bn4k_branch2c          (BatchNorm)
res4l_branch2a         (Conv2D)
bn4l_branch2a          (BatchNorm)
res4l_branch2b         (Conv2D)
bn4l_branch2b          (BatchNorm)
res4l_branch2c         (Conv2D)
bn4l_branch2c          (BatchNorm)
res4m_branch2a         (Conv2D)
bn4m_branch2a          (BatchNorm)
res4m_branch2b         (Conv2D)
bn4m_branch2b          (BatchNorm)
res4m_branch2c         (Conv2D)
bn4m_branch2c          (BatchNorm)
res4n_branch2a         (Conv2D)
bn4n_branch2a          (BatchNorm)
res4n_branch2b         (Conv2D)
bn4n_branch2b          (BatchNorm)
res4n_branch2c         (Conv2D)
bn4n_branch2c          (BatchNorm)
res4o_branch2a         (Conv2D)
bn4o_branch2a          (BatchNorm)
res4o_branch2b         (Conv2D)
bn4o_branch2b          (BatchNorm)
res4o_branch2c         (Conv2D)
bn4o_branch2c          (BatchNorm)
res4p_branch2a         (Conv2D)
bn4p_branch2a          (BatchNorm)
res4p_branch2b         (Conv2D)
bn4p_branch2b          (BatchNorm)
res4p_branch2c         (Conv2D)
bn4p_branch2c          (BatchNorm)
res4q_branch2a         (Conv2D)
bn4q_branch2a          (BatchNorm)
res4q_branch2b         (Conv2D)
bn4q_branch2b          (BatchNorm)
res4q_branch2c         (Conv2D)
bn4q_branch2c          (BatchNorm)
res4r_branch2a         (Conv2D)
bn4r_branch2a          (BatchNorm)
```

```
bn4r_branch2a          (BatchNorm)
res4r_branch2b         (Conv2D)
bn4r_branch2b          (BatchNorm)
res4r_branch2c         (Conv2D)
bn4r_branch2c          (BatchNorm)
res4s_branch2a         (Conv2D)
bn4s_branch2a          (BatchNorm)
res4s_branch2b         (Conv2D)
bn4s_branch2b          (BatchNorm)
res4s_branch2c         (Conv2D)
bn4s_branch2c          (BatchNorm)
res4t_branch2a         (Conv2D)
bn4t_branch2a          (BatchNorm)
res4t_branch2b         (Conv2D)
bn4t_branch2b          (BatchNorm)
res4t_branch2c         (Conv2D)
bn4t_branch2c          (BatchNorm)
res4u_branch2a         (Conv2D)
bn4u_branch2a          (BatchNorm)
res4u_branch2b         (Conv2D)
bn4u_branch2b          (BatchNorm)
res4u_branch2c         (Conv2D)
bn4u_branch2c          (BatchNorm)
res4v_branch2a         (Conv2D)
bn4v_branch2a          (BatchNorm)
res4v_branch2b         (Conv2D)
bn4v_branch2b          (BatchNorm)
res4v_branch2c         (Conv2D)
bn4v_branch2c          (BatchNorm)
res4w_branch2a         (Conv2D)
bn4w_branch2a          (BatchNorm)
res4w_branch2b         (Conv2D)
bn4w_branch2b          (BatchNorm)
res4w_branch2c         (Conv2D)
bn4w_branch2c          (BatchNorm)
res5a_branch2a         (Conv2D)
bn5a_branch2a          (BatchNorm)
res5a_branch2b         (Conv2D)
bn5a_branch2b          (BatchNorm)
res5a_branch2c         (Conv2D)
res5a_branch1          (Conv2D)
bn5a_branch2c          (BatchNorm)
bn5a_branch1           (BatchNorm)
res5b_branch2a         (Conv2D)
bn5b_branch2a          (BatchNorm)
res5b_branch2b         (Conv2D)
bn5b_branch2b          (BatchNorm)
res5b_branch2c         (Conv2D)
bn5b_branch2c          (BatchNorm)
res5c_branch2a         (Conv2D)
bn5c_branch2a          (BatchNorm)
res5c_branch2b         (Conv2D)
bn5c_branch2b          (BatchNorm)
res5c_branch2c         (Conv2D)
bn5c_branch2c          (BatchNorm)
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_conv4       (TimeDistributed)
mrcnn_mask_bn4         (TimeDistributed)
mrcnn_bbox_fc          (TimeDistributed)
mrcnn_mask_deconv      (TimeDistributed)
mrcnn_class_logits     (TimeDistributed)
mrcnn_mask             (TimeDistributed)
Epoch 51/100
300/300 [==============================] - 173s 577ms/step - loss: 2.2454 - rpn_class_loss: 0.0691
- rpn_bbox_loss: 0.6904 - mrcnn_class_loss: 0.6378 - mrcnn_bbox_loss: 0.4355 - mrcnn_mask_loss: 0.
4126 - val_loss: 2.6560 - val_rpn_class_loss: 0.0705 - val_rpn_bbox_loss: 0.8033 -
val_mrcnn_class_loss: 0.8899 - val_mrcnn_bbox_loss: 0.4848 - val_mrcnn_mask_loss: 0.4075
Epoch 52/100
300/300 [==============================] - 134s 446ms/step - loss: 1.7878 - rpn_class_loss: 0.0472
- rpn_bbox_loss: 0.5557 - mrcnn_class_loss: 0.4814 - mrcnn_bbox_loss: 0.3352 - mrcnn_mask_loss: 0.
3682 - val_loss: 2.4269 - val_rpn_class_loss: 0.0493 - val_rpn_bbox_loss: 0.7802 -
val_mrcnn_class_loss: 0.6485 - val_mrcnn_bbox_loss: 0.4995 - val_mrcnn_mask_loss: 0.4494
Epoch 53/100
300/300 [==============================] - 133s 443ms/step - loss: 1.8420 - rpn_class_loss: 0.0521
- rpn_bbox_loss: 0.5684 - mrcnn_class_loss: 0.5263 - mrcnn_bbox_loss: 0.3295 - mrcnn_mask_loss: 0.
3657 - val_loss: 3.0404 - val_rpn_class_loss: 0.0763 - val_rpn_bbox_loss: 1.3315 -
val_mrcnn_class_loss: 0.6312 - val_mrcnn_bbox_loss: 0.5300 - val_mrcnn_mask_loss: 0.4714
Epoch 54/100
300/300 [==============================] - 133s 445ms/step - loss: 2.0507 - rpn_class_loss: 0.0577
- rpn_bbox_loss: 0.8005 - mrcnn_class_loss: 0.5003 - mrcnn_bbox_loss: 0.3405 - mrcnn_mask_loss: 0.
3517 - val_loss: 3.0364 - val_rpn_class_loss: 0.0743 - val_rpn_bbox_loss: 1.1570 -
val_mrcnn_class_loss: 0.8528 - val_mrcnn_bbox_loss: 0.5308 - val_mrcnn_mask_loss: 0.4215
Epoch 55/100
300/300 [==============================] - 133s 445ms/step - loss: 1.8375 - rpn_class_loss: 0.0436
- rpn_bbox_loss: 0.5914 - mrcnn_class_loss: 0.4851 - mrcnn_bbox_loss: 0.3481 - mrcnn_mask_loss: 0.
3693 - val_loss: 2.2341 - val_rpn_class_loss: 0.0503 - val_rpn_bbox_loss: 0.8458 -
val_mrcnn_class_loss: 0.5456 - val_mrcnn_bbox_loss: 0.3761 - val_mrcnn_mask_loss: 0.4162
Epoch 56/100
300/300 [==============================] - 133s 444ms/step - loss: 1.9004 - rpn_class_loss: 0.0665
- rpn_bbox_loss: 0.6691 - mrcnn_class_loss: 0.4574 - mrcnn_bbox_loss: 0.3318 - mrcnn_mask_loss: 0.
3756 - val_loss: 2.5189 - val_rpn_class_loss: 0.0510 - val_rpn_bbox_loss: 0.8996 -
val_mrcnn_class_loss: 0.7023 - val_mrcnn_bbox_loss: 0.4585 - val_mrcnn_mask_loss: 0.4074
Epoch 57/100
300/300 [==============================] - 134s 447ms/step - loss: 1.5519 - rpn_class_loss: 0.0429
- rpn_bbox_loss: 0.5033 - mrcnn_class_loss: 0.4045 - mrcnn_bbox_loss: 0.2754 - mrcnn_mask_loss: 0.
3259 - val_loss: 3.0286 - val_rpn_class_loss: 0.0663 - val_rpn_bbox_loss: 1.0975 -
val_mrcnn_class_loss: 0.9550 - val_mrcnn_bbox_loss: 0.5016 - val_mrcnn_mask_loss: 0.4083
Epoch 58/100
300/300 [==============================] - 133s 443ms/step - loss: 1.6212 - rpn_class_loss: 0.0484
- rpn_bbox_loss: 0.5247 - mrcnn_class_loss: 0.4261 - mrcnn_bbox_loss: 0.2986 - mrcnn_mask_loss: 0.
3234 - val_loss: 2.5640 - val_rpn_class_loss: 0.0385 - val_rpn_bbox_loss: 0.9130 -
val_mrcnn_class_loss: 0.7787 - val_mrcnn_bbox_loss: 0.4198 - val_mrcnn_mask_loss: 0.4140
Epoch 59/100
300/300 [==============================] - 134s 447ms/step - loss: 1.6918 - rpn_class_loss: 0.0485
- rpn_bbox_loss: 0.4776 - mrcnn_class_loss: 0.4987 - mrcnn_bbox_loss: 0.3220 - mrcnn_mask_loss: 0.
3450 - val_loss: 2.4019 - val_rpn_class_loss: 0.0475 - val_rpn_bbox_loss: 0.7810 -
val_mrcnn_class_loss: 0.7568 - val_mrcnn_bbox_loss: 0.4631 - val_mrcnn_mask_loss: 0.3534
Epoch 60/100
300/300 [==============================] - 133s 443ms/step - loss: 1.7917 - rpn_class_loss: 0.0505
- rpn_bbox_loss: 0.6415 - mrcnn_class_loss: 0.4096 - mrcnn_bbox_loss: 0.3232 - mrcnn_mask_loss: 0.
3669 - val_loss: 2.4153 - val_rpn_class_loss: 0.0414 - val_rpn_bbox_loss: 0.8974 -
val_mrcnn_class_loss: 0.6533 - val_mrcnn_bbox_loss: 0.4611 - val_mrcnn_mask_loss: 0.3621
Epoch 61/100
300/300 [==============================] - 134s 447ms/step - loss: 1.6429 - rpn_class_loss: 0.0403
- rpn_bbox_loss: 0.5051 - mrcnn_class_loss: 0.4198 - mrcnn_bbox_loss: 0.3209 - mrcnn_mask_loss: 0.
3568 - val_loss: 2.3016 - val_rpn_class_loss: 0.0426 - val_rpn_bbox_loss: 0.8029 -
val_mrcnn_class_loss: 0.6067 - val_mrcnn_bbox_loss: 0.4313 - val_mrcnn_mask_loss: 0.4181
Epoch 62/100
300/300 [==============================] - 133s 443ms/step - loss: 1.8172 - rpn_class_loss: 0.0514
- rpn_bbox_loss: 0.6818 - mrcnn_class_loss: 0.3964 - mrcnn_bbox_loss: 0.3299 - mrcnn_mask_loss: 0.
3577 - val_loss: 2.3769 - val_rpn_class_loss: 0.0443 - val_rpn_bbox_loss: 0.9325 -
val_mrcnn_class_loss: 0.5955 - val_mrcnn_bbox_loss: 0.4195 - val_mrcnn_mask_loss: 0.3852
Epoch 63/100
300/300 [==============================] - 134s 448ms/step - loss: 1.5515 - rpn_class_loss: 0.0429
- rpn_bbox_loss: 0.4917 - mrcnn_class_loss: 0.3710 - mrcnn_bbox_loss: 0.3054 - mrcnn_mask_loss: 0.
3406 - val_loss: 2.6999 - val_rpn_class_loss: 0.0696 - val_rpn_bbox_loss: 1.1430 -
val_mrcnn_class_loss: 0.5465 - val_mrcnn_bbox_loss: 0.4430 - val_mrcnn_mask_loss: 0.4979
Epoch 64/100
300/300 [==============================] - 134s 445ms/step - loss: 1.4715 - rpn_class_loss: 0.0366
- rpn_bbox_loss: 0.4592 - mrcnn_class_loss: 0.3512 - mrcnn_bbox_loss: 0.2951 - mrcnn_mask_loss: 0.
3294 - val_loss: 2.6613 - val_rpn_class_loss: 0.0344 - val_rpn_bbox_loss: 0.7726 -
val_mrcnn_class_loss: 0.8880 - val_mrcnn_bbox_loss: 0.5312 - val_mrcnn_mask_loss: 0.4352
Epoch 65/100
```

```
300/300 [==============================] - 134s 446ms/step - loss: 1.6136 - rpn_class_loss: 0.0423
- rpn_bbox_loss: 0.4745 - mrcnn_class_loss: 0.4534 - mrcnn_bbox_loss: 0.3163 - mrcnn_mask_loss: 0.
3271 - val_loss: 3.0394 - val_rpn_class_loss: 0.0840 - val_rpn_bbox_loss: 1.1808 -
val_mrcnn_class_loss: 0.7396 - val_mrcnn_bbox_loss: 0.5710 - val_mrcnn_mask_loss: 0.4640
Epoch 66/100
300/300 [==============================] - 133s 444ms/step - loss: 1.5479 - rpn_class_loss: 0.0413
- rpn_bbox_loss: 0.4706 - mrcnn_class_loss: 0.3934 - mrcnn_bbox_loss: 0.3057 - mrcnn_mask_loss: 0.
3369 - val_loss: 3.1498 - val_rpn_class_loss: 0.1071 - val_rpn_bbox_loss: 1.2881 -
val_mrcnn_class_loss: 0.8123 - val_mrcnn_bbox_loss: 0.5058 - val_mrcnn_mask_loss: 0.4364
Epoch 67/100
300/300 [==============================] - 133s 445ms/step - loss: 1.5683 - rpn_class_loss: 0.0396
- rpn_bbox_loss: 0.5210 - mrcnn_class_loss: 0.3957 - mrcnn_bbox_loss: 0.2778 - mrcnn_mask_loss: 0.
3342 - val_loss: 2.9993 - val_rpn_class_loss: 0.0607 - val_rpn_bbox_loss: 1.1773 -
val_mrcnn_class_loss: 0.7518 - val_mrcnn_bbox_loss: 0.5915 - val_mrcnn_mask_loss: 0.4180
Epoch 68/100
300/300 [==============================] - 134s 446ms/step - loss: 1.5603 - rpn_class_loss: 0.0386
- rpn_bbox_loss: 0.5519 - mrcnn_class_loss: 0.3406 - mrcnn_bbox_loss: 0.2952 - mrcnn_mask_loss: 0.
3340 - val_loss: 2.8173 - val_rpn_class_loss: 0.0637 - val_rpn_bbox_loss: 1.1858 -
val_mrcnn_class_loss: 0.6769 - val_mrcnn_bbox_loss: 0.4966 - val_mrcnn_mask_loss: 0.3944
Epoch 69/100
300/300 [==============================] - 133s 444ms/step - loss: 1.5145 - rpn_class_loss: 0.0396
- rpn_bbox_loss: 0.4614 - mrcnn_class_loss: 0.3913 - mrcnn_bbox_loss: 0.2809 - mrcnn_mask_loss: 0.
3413 - val_loss: 2.4883 - val_rpn_class_loss: 0.0588 - val_rpn_bbox_loss: 1.3147 -
val_mrcnn_class_loss: 0.3625 - val_mrcnn_bbox_loss: 0.3867 - val_mrcnn_mask_loss: 0.3656
Epoch 70/100
300/300 [==============================] - 134s 446ms/step - loss: 1.5302 - rpn_class_loss: 0.0405
- rpn_bbox_loss: 0.4672 - mrcnn_class_loss: 0.4238 - mrcnn_bbox_loss: 0.2847 - mrcnn_mask_loss: 0.
3140 - val_loss: 2.7866 - val_rpn_class_loss: 0.0488 - val_rpn_bbox_loss: 1.2353 -
val_mrcnn_class_loss: 0.5567 - val_mrcnn_bbox_loss: 0.4855 - val_mrcnn_mask_loss: 0.4602
Epoch 71/100
300/300 [==============================] - 133s 443ms/step - loss: 1.5515 - rpn_class_loss: 0.0385
- rpn_bbox_loss: 0.4588 - mrcnn_class_loss: 0.4188 - mrcnn_bbox_loss: 0.2982 - mrcnn_mask_loss: 0.
3373 - val_loss: 3.1027 - val_rpn_class_loss: 0.0527 - val_rpn_bbox_loss: 1.6463 -
val_mrcnn_class_loss: 0.5298 - val_mrcnn_bbox_loss: 0.4526 - val_mrcnn_mask_loss: 0.4212
Epoch 72/100
300/300 [==============================] - 134s 447ms/step - loss: 1.6428 - rpn_class_loss: 0.0437
- rpn_bbox_loss: 0.5557 - mrcnn_class_loss: 0.3862 - mrcnn_bbox_loss: 0.3041 - mrcnn_mask_loss: 0.
3532 - val_loss: 3.1917 - val_rpn_class_loss: 0.0620 - val_rpn_bbox_loss: 1.4798 -
val_mrcnn_class_loss: 0.7261 - val_mrcnn_bbox_loss: 0.4736 - val_mrcnn_mask_loss: 0.4503
Epoch 73/100
300/300 [==============================] - 133s 444ms/step - loss: 1.6604 - rpn_class_loss: 0.0473
- rpn_bbox_loss: 0.5716 - mrcnn_class_loss: 0.3986 - mrcnn_bbox_loss: 0.3109 - mrcnn_mask_loss: 0.
3319 - val_loss: 2.5500 - val_rpn_class_loss: 0.0453 - val_rpn_bbox_loss: 1.0633 -
val_mrcnn_class_loss: 0.5439 - val_mrcnn_bbox_loss: 0.4489 - val_mrcnn_mask_loss: 0.4486
Epoch 74/100
300/300 [==============================] - 134s 446ms/step - loss: 1.8207 - rpn_class_loss: 0.0546
- rpn_bbox_loss: 0.6259 - mrcnn_class_loss: 0.4371 - mrcnn_bbox_loss: 0.3513 - mrcnn_mask_loss: 0.
3517 - val_loss: 2.5482 - val_rpn_class_loss: 0.0442 - val_rpn_bbox_loss: 1.1662 -
val_mrcnn_class_loss: 0.5212 - val_mrcnn_bbox_loss: 0.4465 - val_mrcnn_mask_loss: 0.3700
Epoch 75/100
300/300 [==============================] - 133s 444ms/step - loss: 1.6938 - rpn_class_loss: 0.0458
- rpn_bbox_loss: 0.5586 - mrcnn_class_loss: 0.4017 - mrcnn_bbox_loss: 0.3406 - mrcnn_mask_loss: 0.
3471 - val_loss: 3.2426 - val_rpn_class_loss: 0.0607 - val_rpn_bbox_loss: 1.3470 -
val_mrcnn_class_loss: 0.8562 - val_mrcnn_bbox_loss: 0.5593 - val_mrcnn_mask_loss: 0.4194
Epoch 76/100
300/300 [==============================] - 134s 446ms/step - loss: 1.5838 - rpn_class_loss: 0.0418
- rpn_bbox_loss: 0.4686 - mrcnn_class_loss: 0.4268 - mrcnn_bbox_loss: 0.3023 - mrcnn_mask_loss: 0.
3444 - val_loss: 2.4246 - val_rpn_class_loss: 0.0376 - val_rpn_bbox_loss: 0.8072 -
val_mrcnn_class_loss: 0.7414 - val_mrcnn_bbox_loss: 0.4713 - val_mrcnn_mask_loss: 0.3671
Epoch 77/100
300/300 [==============================] - 133s 443ms/step - loss: 1.6240 - rpn_class_loss: 0.0461
- rpn_bbox_loss: 0.5355 - mrcnn_class_loss: 0.4017 - mrcnn_bbox_loss: 0.3127 - mrcnn_mask_loss: 0.
3280 - val_loss: 2.9104 - val_rpn_class_loss: 0.0443 - val_rpn_bbox_loss: 1.3117 -
val_mrcnn_class_loss: 0.6763 - val_mrcnn_bbox_loss: 0.4743 - val_mrcnn_mask_loss: 0.4037
Epoch 78/100
300/300 [==============================] - 134s 447ms/step - loss: 1.8669 - rpn_class_loss: 0.0463
- rpn_bbox_loss: 0.5865 - mrcnn_class_loss: 0.5069 - mrcnn_bbox_loss: 0.3607 - mrcnn_mask_loss: 0.
3665 - val_loss: 2.6099 - val_rpn_class_loss: 0.0415 - val_rpn_bbox_loss: 0.7950 -
val_mrcnn_class_loss: 0.8788 - val_mrcnn_bbox_loss: 0.4657 - val_mrcnn_mask_loss: 0.4290
Epoch 79/100
300/300 [==============================] - 133s 444ms/step - loss: 1.5874 - rpn_class_loss: 0.0353
- rpn_bbox_loss: 0.4937 - mrcnn_class_loss: 0.4107 - mrcnn_bbox_loss: 0.2951 - mrcnn_mask_loss: 0.
3527 - val_loss: 2.7627 - val_rpn_class_loss: 0.0552 - val_rpn_bbox_loss: 1.0246 -
val_mrcnn_class_loss: 0.7829 - val_mrcnn_bbox_loss: 0.4633 - val_mrcnn_mask_loss: 0.4368
Epoch 80/100
300/300 [==============================] - 133s 444ms/step - loss: 1.7732 - rpn_class_loss: 0.0496
- rpn_bbox_loss: 0.6283 - mrcnn_class_loss: 0.4323 - mrcnn_bbox_loss: 0.3131 - mrcnn_mask_loss: 0.
```

```
3498 - val_loss: 2.8438 - val_rpn_class_loss: 0.0546 - val_rpn_bbox_loss: 1.1645 -
val_mrcnn_class_loss: 0.7541 - val_mrcnn_bbox_loss: 0.4724 - val_mrcnn_mask_loss: 0.3982
Epoch 81/100
300/300 [==============================] - 134s 447ms/step - loss: 1.6286 - rpn_class_loss: 0.0503
- rpn_bbox_loss: 0.5116 - mrcnn_class_loss: 0.4177 - mrcnn_bbox_loss: 0.3056 - mrcnn_mask_loss: 0.
3434 - val_loss: 3.1482 - val_rpn_class_loss: 0.0675 - val_rpn_bbox_loss: 1.5108 -
val_mrcnn_class_loss: 0.6187 - val_mrcnn_bbox_loss: 0.4980 - val_mrcnn_mask_loss: 0.4531
Epoch 82/100
300/300 [==============================] - 133s 444ms/step - loss: 1.6641 - rpn_class_loss: 0.0375
- rpn_bbox_loss: 0.6041 - mrcnn_class_loss: 0.3895 - mrcnn_bbox_loss: 0.3014 - mrcnn_mask_loss: 0.
3316 - val_loss: 2.7245 - val_rpn_class_loss: 0.0713 - val_rpn_bbox_loss: 1.1085 -
val_mrcnn_class_loss: 0.7042 - val_mrcnn_bbox_loss: 0.4419 - val_mrcnn_mask_loss: 0.3987
Epoch 83/100
300/300 [==============================] - 134s 446ms/step - loss: 1.5502 - rpn_class_loss: 0.0551
- rpn_bbox_loss: 0.5168 - mrcnn_class_loss: 0.3512 - mrcnn_bbox_loss: 0.2871 - mrcnn_mask_loss: 0.
3400 - val_loss: 2.3537 - val_rpn_class_loss: 0.0471 - val_rpn_bbox_loss: 0.8273 -
val_mrcnn_class_loss: 0.6905 - val_mrcnn_bbox_loss: 0.4050 - val_mrcnn_mask_loss: 0.3837
Epoch 84/100
300/300 [==============================] - 134s 446ms/step - loss: 1.5527 - rpn_class_loss: 0.0348
- rpn_bbox_loss: 0.5187 - mrcnn_class_loss: 0.3514 - mrcnn_bbox_loss: 0.3100 - mrcnn_mask_loss: 0.
3378 - val_loss: 2.8964 - val_rpn_class_loss: 0.0704 - val_rpn_bbox_loss: 1.0056 -
val_mrcnn_class_loss: 0.9262 - val_mrcnn_bbox_loss: 0.4685 - val_mrcnn_mask_loss: 0.4256
Epoch 85/100
300/300 [==============================] - 134s 446ms/step - loss: 1.6330 - rpn_class_loss: 0.0512
- rpn_bbox_loss: 0.5464 - mrcnn_class_loss: 0.4016 - mrcnn_bbox_loss: 0.3011 - mrcnn_mask_loss: 0.
3326 - val_loss: 2.7338 - val_rpn_class_loss: 0.0601 - val_rpn_bbox_loss: 0.8718 -
val_mrcnn_class_loss: 0.9936 - val_mrcnn_bbox_loss: 0.4081 - val_mrcnn_mask_loss: 0.4003
Epoch 86/100
300/300 [==============================] - 134s 446ms/step - loss: 1.6275 - rpn_class_loss: 0.0483
- rpn_bbox_loss: 0.5643 - mrcnn_class_loss: 0.3758 - mrcnn_bbox_loss: 0.3008 - mrcnn_mask_loss: 0.
3383 - val_loss: 2.8594 - val_rpn_class_loss: 0.0529 - val_rpn_bbox_loss: 1.1169 -
val_mrcnn_class_loss: 0.6938 - val_mrcnn_bbox_loss: 0.5146 - val_mrcnn_mask_loss: 0.4811
Epoch 87/100
300/300 [==============================] - 134s 446ms/step - loss: 1.6912 - rpn_class_loss: 0.0459
- rpn_bbox_loss: 0.5366 - mrcnn_class_loss: 0.4365 - mrcnn_bbox_loss: 0.3042 - mrcnn_mask_loss: 0.
3680 - val_loss: 3.1277 - val_rpn_class_loss: 0.0882 - val_rpn_bbox_loss: 1.4279 -
val_mrcnn_class_loss: 0.6515 - val_mrcnn_bbox_loss: 0.5396 - val_mrcnn_mask_loss: 0.4205
Epoch 88/100
300/300 [==============================] - 133s 443ms/step - loss: 1.5363 - rpn_class_loss: 0.0426
- rpn_bbox_loss: 0.5169 - mrcnn_class_loss: 0.3713 - mrcnn_bbox_loss: 0.2821 - mrcnn_mask_loss: 0.
3234 - val_loss: 2.6225 - val_rpn_class_loss: 0.0654 - val_rpn_bbox_loss: 0.9418 -
val_mrcnn_class_loss: 0.6669 - val_mrcnn_bbox_loss: 0.5195 - val_mrcnn_mask_loss: 0.4289
Epoch 89/100
300/300 [==============================] - 134s 445ms/step - loss: 1.4369 - rpn_class_loss: 0.0370
- rpn_bbox_loss: 0.4169 - mrcnn_class_loss: 0.3557 - mrcnn_bbox_loss: 0.2903 - mrcnn_mask_loss: 0.
3370 - val_loss: 2.5942 - val_rpn_class_loss: 0.0401 - val_rpn_bbox_loss: 0.9923 -
val_mrcnn_class_loss: 0.7119 - val_mrcnn_bbox_loss: 0.4204 - val_mrcnn_mask_loss: 0.4295
Epoch 90/100
300/300 [==============================] - 133s 444ms/step - loss: 1.5362 - rpn_class_loss: 0.0394
- rpn_bbox_loss: 0.4638 - mrcnn_class_loss: 0.3858 - mrcnn_bbox_loss: 0.2978 - mrcnn_mask_loss: 0.
3493 - val_loss: 2.3472 - val_rpn_class_loss: 0.0387 - val_rpn_bbox_loss: 0.8150 -
val_mrcnn_class_loss: 0.6891 - val_mrcnn_bbox_loss: 0.4114 - val_mrcnn_mask_loss: 0.3930
Epoch 91/100
300/300 [==============================] - 134s 447ms/step - loss: 1.5472 - rpn_class_loss: 0.0389
- rpn_bbox_loss: 0.5549 - mrcnn_class_loss: 0.3371 - mrcnn_bbox_loss: 0.2767 - mrcnn_mask_loss: 0.
3396 - val_loss: 2.5220 - val_rpn_class_loss: 0.0368 - val_rpn_bbox_loss: 0.9982 -
val_mrcnn_class_loss: 0.5877 - val_mrcnn_bbox_loss: 0.4823 - val_mrcnn_mask_loss: 0.4170
Epoch 92/100
300/300 [==============================] - 133s 444ms/step - loss: 1.5038 - rpn_class_loss: 0.0435
- rpn_bbox_loss: 0.4902 - mrcnn_class_loss: 0.3780 - mrcnn_bbox_loss: 0.2694 - mrcnn_mask_loss: 0.
3227 - val_loss: 2.6678 - val_rpn_class_loss: 0.0530 - val_rpn_bbox_loss: 1.3003 -
val_mrcnn_class_loss: 0.3533 - val_mrcnn_bbox_loss: 0.5440 - val_mrcnn_mask_loss: 0.4172
Epoch 93/100
300/300 [==============================] - 133s 443ms/step - loss: 1.4114 - rpn_class_loss: 0.0390
- rpn_bbox_loss: 0.4234 - mrcnn_class_loss: 0.3408 - mrcnn_bbox_loss: 0.2797 - mrcnn_mask_loss: 0.
3285 - val_loss: 3.1741 - val_rpn_class_loss: 0.0588 - val_rpn_bbox_loss: 1.4503 -
val_mrcnn_class_loss: 0.7234 - val_mrcnn_bbox_loss: 0.5374 - val_mrcnn_mask_loss: 0.4042
Epoch 94/100
300/300 [==============================] - 134s 448ms/step - loss: 1.5587 - rpn_class_loss: 0.0437
- rpn_bbox_loss: 0.4823 - mrcnn_class_loss: 0.3906 - mrcnn_bbox_loss: 0.2922 - mrcnn_mask_loss: 0.
3499 - val_loss: 2.4744 - val_rpn_class_loss: 0.0443 - val_rpn_bbox_loss: 0.9599 -
val_mrcnn_class_loss: 0.6020 - val_mrcnn_bbox_loss: 0.4620 - val_mrcnn_mask_loss: 0.4062
Epoch 95/100
300/300 [==============================] - 133s 444ms/step - loss: 1.7295 - rpn_class_loss: 0.0455
- rpn_bbox_loss: 0.6497 - mrcnn_class_loss: 0.3751 - mrcnn_bbox_loss: 0.3066 - mrcnn_mask_loss: 0.
3526 - val_loss: 2.7338 - val_rpn_class_loss: 0.0699 - val_rpn_bbox_loss: 0.9884 -
val_mrcnn_class_loss: 0.6995 - val_mrcnn_bbox_loss: 0.4979 - val_mrcnn_mask_loss: 0.4781
```

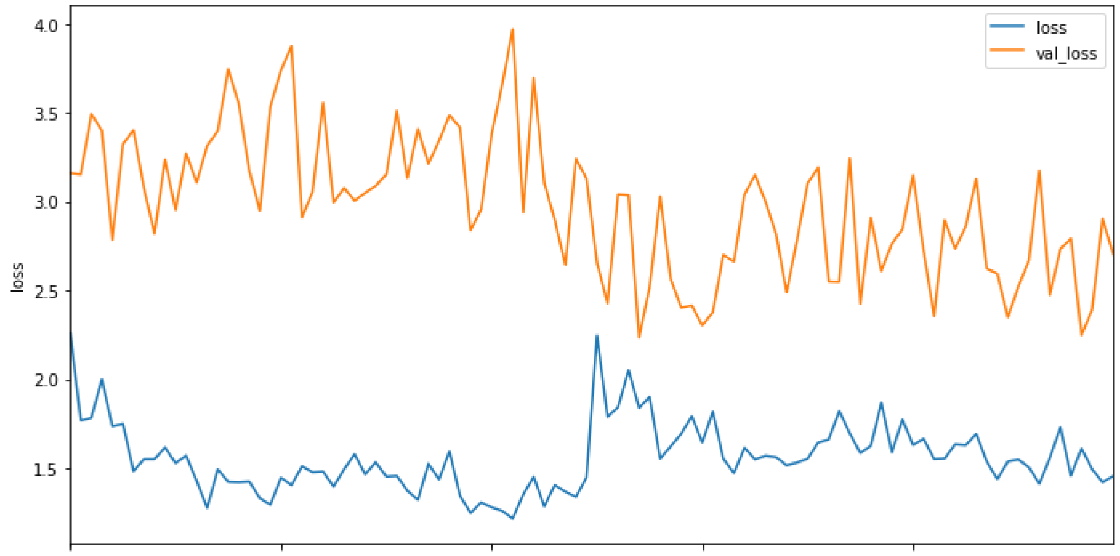
```
Epoch 96/100
300/300 [==============================] - 131s 438ms/step - loss: 1.4570 - rpn_class_loss: 0.0416
- rpn_bbox_loss: 0.5279 - mrcnn_class_loss: 0.3225 - mrcnn_bbox_loss: 0.2445 - mrcnn_mask_loss: 0.
3205 - val_loss: 2.7907 - val_rpn_class_loss: 0.0626 - val_rpn_bbox_loss: 1.1466 -
val_mrcnn_class_loss: 0.7020 - val_mrcnn_bbox_loss: 0.4854 - val_mrcnn_mask_loss: 0.3940
Epoch 97/100
300/300 [==============================] - 133s 443ms/step - loss: 1.6091 - rpn_class_loss: 0.0414
- rpn_bbox_loss: 0.5043 - mrcnn_class_loss: 0.3957 - mrcnn_bbox_loss: 0.3147 - mrcnn_mask_loss: 0.
3530 - val_loss: 2.2484 - val_rpn_class_loss: 0.0634 - val_rpn_bbox_loss: 0.8075 -
val_mrcnn_class_loss: 0.5892 - val_mrcnn_bbox_loss: 0.4345 - val_mrcnn_mask_loss: 0.3538
Epoch 98/100
300/300 [==============================] - 133s 443ms/step - loss: 1.4933 - rpn_class_loss: 0.0357
- rpn_bbox_loss: 0.4558 - mrcnn_class_loss: 0.3525 - mrcnn_bbox_loss: 0.2971 - mrcnn_mask_loss: 0.
3522 - val_loss: 2.3911 - val_rpn_class_loss: 0.0409 - val_rpn_bbox_loss: 0.7600 -
val_mrcnn_class_loss: 0.7498 - val_mrcnn_bbox_loss: 0.4901 - val_mrcnn_mask_loss: 0.3504
Epoch 99/100
300/300 [==============================] - 134s 446ms/step - loss: 1.4196 - rpn_class_loss: 0.0336
- rpn_bbox_loss: 0.4153 - mrcnn_class_loss: 0.3606 - mrcnn_bbox_loss: 0.2765 - mrcnn_mask_loss: 0.
3336 - val_loss: 2.9038 - val_rpn_class_loss: 0.0408 - val_rpn_bbox_loss: 1.0689 -
val_mrcnn_class_loss: 0.8462 - val_mrcnn_bbox_loss: 0.5308 - val_mrcnn_mask_loss: 0.4170
Epoch 100/100
300/300 [==============================] - 133s 443ms/step - loss: 1.4545 - rpn_class_loss: 0.0424
- rpn_bbox_loss: 0.4258 - mrcnn_class_loss: 0.3800 - mrcnn_bbox_loss: 0.2678 - mrcnn_mask_loss: 0.
3384 - val_loss: 2.7036 - val_rpn_class_loss: 0.0532 - val_rpn_bbox_loss: 0.9969 -
val_mrcnn_class_loss: 0.7799 - val_mrcnn_bbox_loss: 0.4671 - val_mrcnn_mask_loss: 0.4065
```

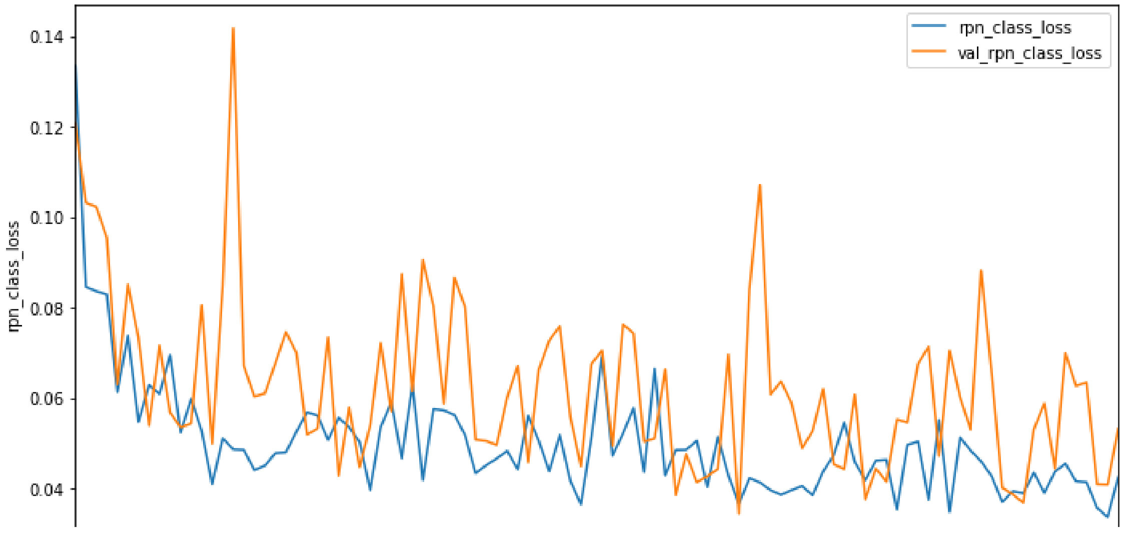
In [15]:

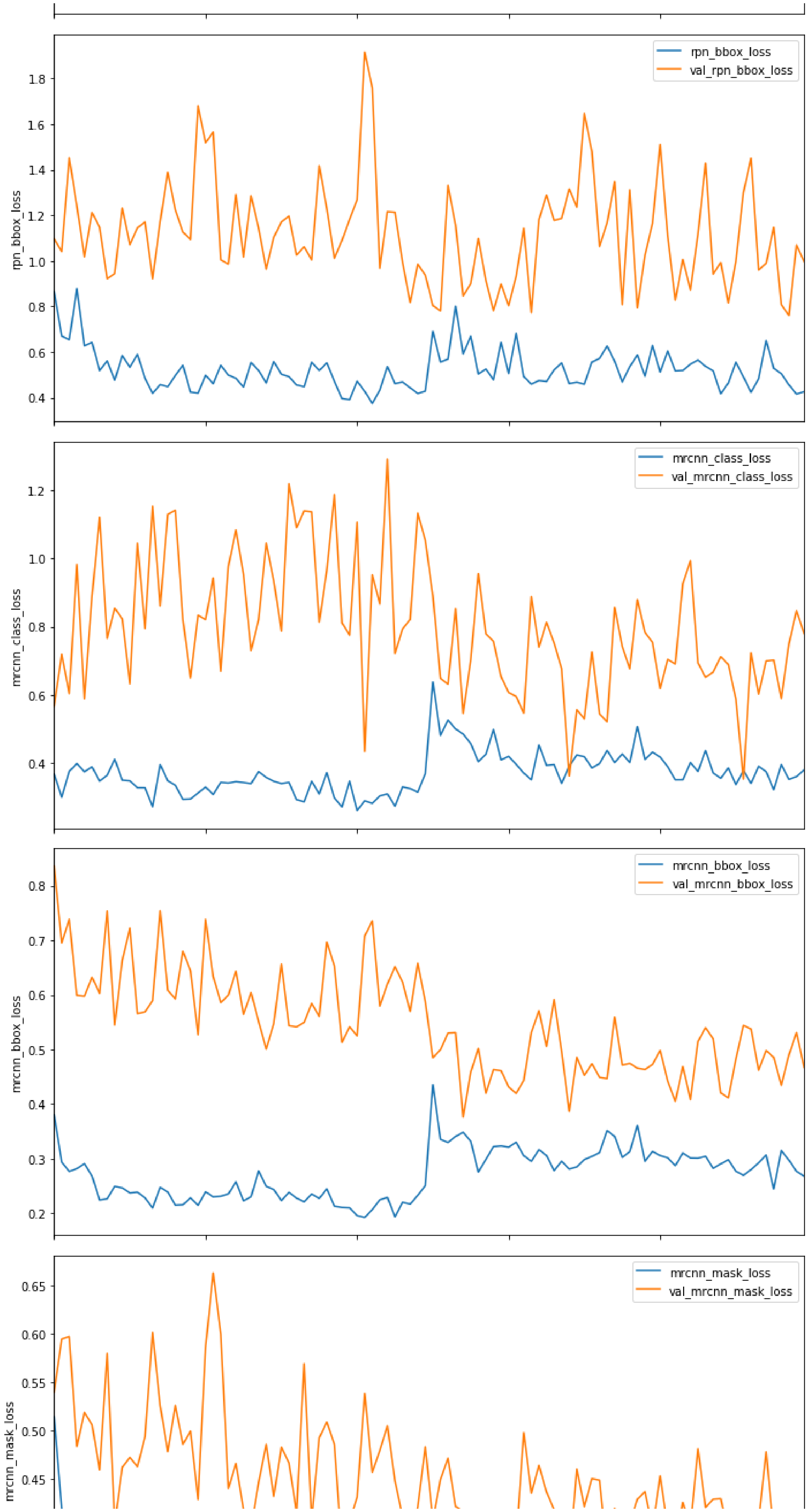
```
history_fine = get_history(model)
```

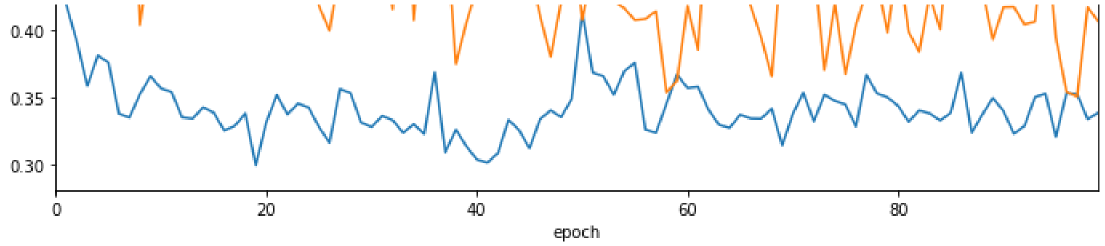
In [16]:

```
histplot(history_fine)
```






rpn_bbox_loss
val_rpn_bbox_loss
rpn_bbox_loss
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
mrcnn_class_loss
val_mrcnn_class_loss
mrcnn_class_loss
1.2
1.0
0.8
0.6
0.4
mrcnn_bbox_loss
val_mrcnn_bbox_loss
mrcnn_bbox_loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
mrcnn_mask_loss
val_mrcnn_mask_loss
mrcnn_mask_loss
0.65
0.60
0.55
0.50
0.45



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 [27]:

```
ls /gpfs01/home/ppzsb1/astobjdet/Mask_RCNN/logs/
```

```
/usr/bin/sh: module: line 1: syntax error: unexpected end of file
/usr/bin/sh: error importing function definition for `BASH_FUNC_module'
/usr/bin/sh: ml: line 1: syntax error: unexpected end of file
/usr/bin/sh: error importing function definition for `BASH_FUNC_ml'
astroobjects20190401T2254/
astroobjects20190401T2341/
astroobjects20190401T2343/
astroobjects20190402T0007/
astroobjects20190402T0009/
astroobjects20190409T1641/
astroobjects20190412T1131/
astroobjects20190412T1207/
astroobjects20190412T1210/
astroobjects20190412T1212/
astroobjects20190412T1216/
astroobjects20190412T1223/
astroobjects20190416T1332/
astroobjects20190416T1346/
astroobjects20190420T1532/
astroobjects20190420T1541/
astroobjects20190422T2042/
astroobjects20190424T1422/
astroobjects20190424T1436/
astroobjects20190425T1230/
colourconcmag20190506T2044/
config_crops_m-0.05_s0.5_q820190510T1600/
config_crops_m0.05_s0.5_q820190510T1602/
config_crops_m0_s0.1_q820190512T1738/
config_crops_m0_s0.3_q820190512T1740/
config_crops_m0_s0.7_q820190512T2215/
config_crops_m0_s0.9_q820190512T2216/
crops_standard_m0_s0.5_q8_standardmodel20190509T1933/
```

In [10]:

```
class InferenceConfig(AstroObjectsConfig):
    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
model_path = os.path.join(ROOT_DIR,
"logs/crops_standard_m0_s0.5_q8_standardmodel20190509T1933/mask_rcnn_crops_standard_m0_s0.5_q8_star
model_0100.h5")
#model_path = model.find_last()

# Load trained weights
print("Loading weights from ", model_path)
model.load_weights(model_path, by_name=True)
```

```
WARNING:tensorflow:From /gpfs01/home/ppzsb1/.conda/envs/astobjdet/lib/python3.6/site-
packages/tensorflow/python/ops/sparse_ops.py:1165: sparse_to_dense (from
tensorflow.python.ops.sparse_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Create a `tf.sparse.SparseTensor` and use `tf.sparse.to_dense` instead.
Loading weights from
/gpfs01/home/ppzsb1/astobjdet/Mask_RCNN/logs/crops_standard_m0_s0.5_q8_standardmodel20190509T1933/m
rcnn_crops_standard_m0_s0.5_q8_standardmodel_0100.h5
```

Use randomly selected image to visually compare model to masks

In [12]:

```
# Test on a random image
#image_id = random.choice(dataset_val.image_ids)
# some interesting images:
image_id = 956
#image_id = 608
#image_id = 741
#image_id = 1729
#image_id = 453
#image_id = 446

original_image, image_meta, gt_class_id, gt_bbox, gt_mask =\
    modellib.load_image_gt(dataset_val, inference_config,
                           image_id, use_mini_mask=False)
fig = plt.figure(figsize=(8, 8))
plt.imshow(original_image);
# change name each save..
from matplotlib.image import imsave
fig.savefig('m0_s0-5_q8_standardmodel.png')
```

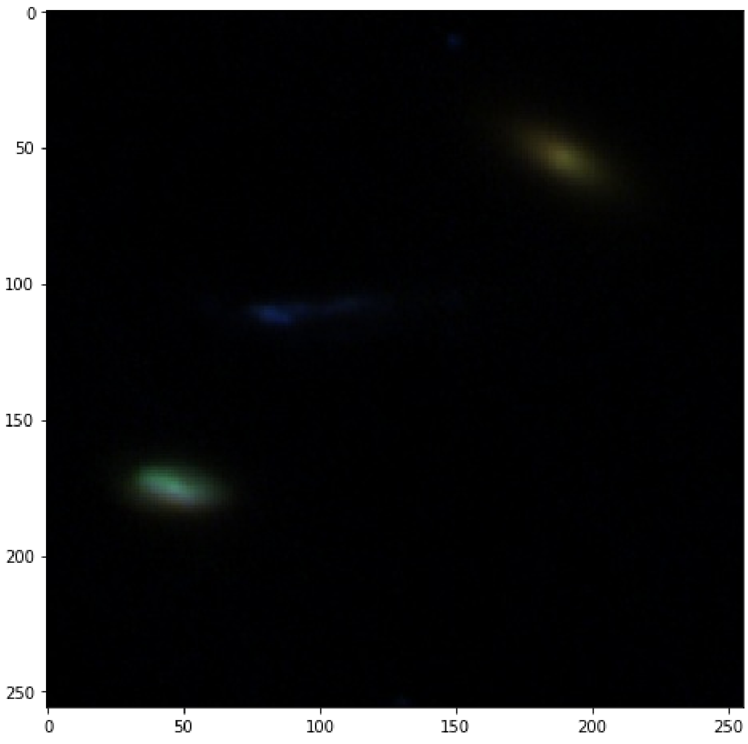


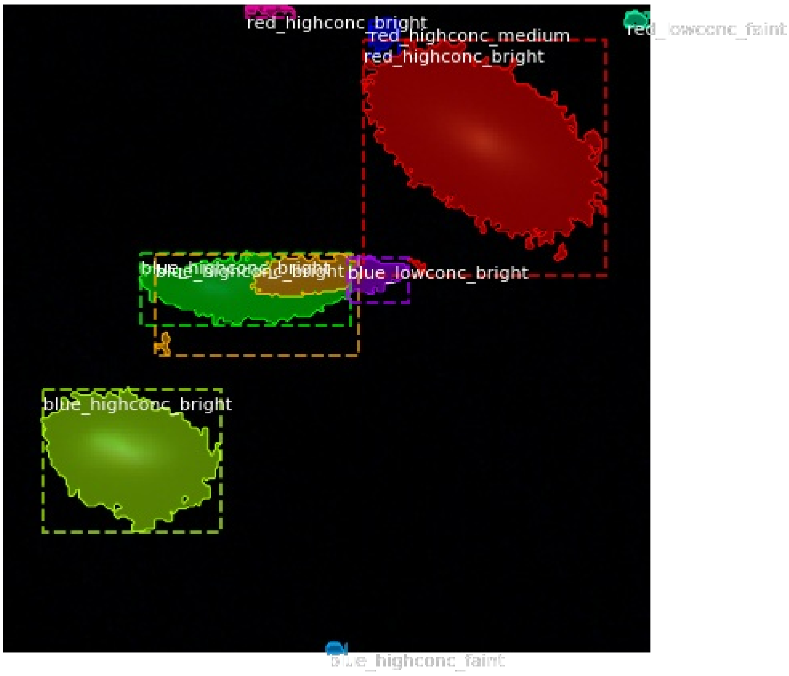
Image with original segmentation

Image with original segmentation

In [13]:

```
log("original_image", original_image)
log("image_meta", image_meta)
log("gt_class_id", gt_class_id)
log("gt_bbox", gt_bbox)
log("gt_mask", gt_mask)
from PIL import Image
fig, ax = plt.subplots(figsize=(8, 8))
visualize.display_instances(original_image, gt_bbox, gt_mask, gt_class_id,
                            dataset_train.class_names, ax=ax)
#Image.save('m0_s0-5_q8_standardmodelSEx')
#fig.save("m0_s0-5_q8_standardmodelSEx", "PNG")
fig.savefig('orig_seg_standard_model.png')
```

```
original_image           shape: (256, 256, 3)         min:    0.00000  max:  134.00000  uint8
image_meta               shape: (25,)                 min:    0.00000  max:  956.00000  int64
gt_class_id              shape: (9,)                  min:    3.00000  max:   12.00000  int32
gt_bbox                  shape: (9, 4)                min:    0.00000  max:  256.00000  int32
gt_mask                  shape: (256, 256, 9)         min:    0.00000  max:    1.00000  bool
```



In [96]:

```
pwd
```

Out[96]:

```
'/gpfs01/home/ppzsb1/astobjdet/Mask_RCNN/samples/AstroDetector/Model Notes'
```

Image with Mask R-CNN objects

In [14]:

```
# Red, high concentration implies Elipitical Galaxy
# Blue, low concentration implies Spiral Galaxy
dataset_val.class_names[1] = "Faint Spiral Galaxy"
dataset_val.class_names[2] = "Medium Spiral Galaxy"
dataset_val.class_names[3] = "Bright Spiral Galaxy"
dataset_val.class_names[10] = "Faint Eliptical Galaxy"
dataset_val.class_names[11] = "Medium Eliptical Galaxy"
dataset_val.class_names[12] = "Bright Eliptical Galaxy"
dataset_val.class_names
```

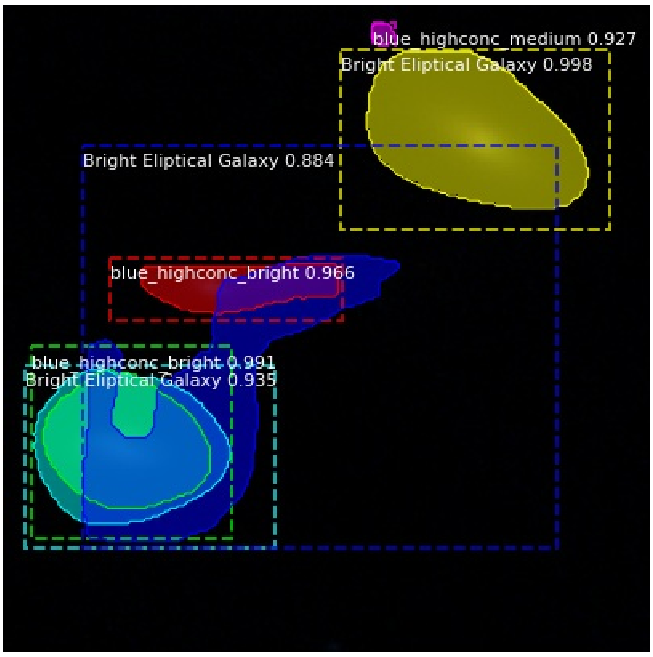
Out[14]:

```
['BG',
 'Faint Spiral Galaxy',
 'Medium Spiral Galaxy',
 'Bright Spiral Galaxy',
 'blue_highconc_faint',
 'blue_highconc_medium',
 'blue_highconc_bright',
 'red_lowconc_faint',
 'red_lowconc_medium',
 'red_lowconc_bright',
 'Faint Eliptical Galaxy',
 'Medium Eliptical Galaxy',
 'Bright Eliptical Galaxy']
```

In [16]:

```
results = model.detect([original_image], verbose=1)

r = results[0]
fig, ax = plt.subplots(figsize=(8, 8))
visualize.display_instances(original_image, r['rois'], r['masks'], r['class_ids'],
                            dataset_val.class_names, r['scores'], ax=ax)

fig.savefig('ourmodel_seg_standard_model.png')
```

```
Processing 1 images
image                    shape: (256, 256, 3)         min:    0.00000  max:  134.00000  uint8
molded_images            shape: (1, 256, 256, 3)      min: -123.70000  max:   17.20000  float64
image_metas              shape: (1, 25)               min:    0.00000  max:  256.00000  int64
anchors                  shape: (1, 16368, 4)         min:   -0.35494  max:    1.10396  float32
```



In [69]:

```
r['masks'].shape
```

Out[69]:

```
(256, 256, 6)
```

In [17]:

```
IoU = utils.compute_overlaps_masks(gt_mask, r['masks'])
print(IoU)
```

```
[[0.         0.         0.         0.         0.5619048  0.        ]
 [0.7809137  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.        ]
 [0.         0.7723139  0.         0.68036777 0.         0.28240317]
 [0.         0.         0.49240506 0.         0.         0.08461419]
 [0.         0.         0.2854985  0.         0.         0.08094262]
 [0.         0.         0.         0.         0.         0.03060875]
 [0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.        ]]
```

In [18]:

```
# how many result objects overlap with each GT object
n_good_overlap = (IoU > 0.5).sum(axis=1)
n_good_overlap
```

Out[18]:

```
array([1, 1, 0, 2, 0, 0, 0, 0, 0])
```

In [19]:

```
# How many GT objects have a detection (even if more than one)
(n_good_overlap > 0).sum()
```

Out[19]:

```
3
```

In [20]:

```
# How many GT objects have multiple detections
(n_good_overlap > 1).sum()
```

Out[20]:

```
1
```

Evaluation

Evaluation method 1 - mean Average Precision (mAP)

In [21]:

```
# Compute VOC-Style mAP @ IoU=0.5
# Running on 100 images. Increase for better accuracy.
image_ids = np.random.choice(dataset_val.image_ids, 150)
#image_ids = dataset_val.image_ids
APs = []
IoU = []
for image_id in image_ids:
    # Load image and ground truth data
    image, image_meta, gt_class_id, gt_bbox, gt_mask =\
        modellib.load_image_gt(dataset_val, inference_config,
                               image_id, use_mini_mask=False)
    molded_images = np.expand_dims(modellib.mold_image(image, inference_config), 0)
    # Run object detection
    results = model.detect([image], verbose=0)
    r = results[0]
    # Compute AP
    AP, precisions, recalls, overlaps =\
        utils.compute_ap(gt_bbox, gt_class_id, gt_mask,
                         r["rois"], r["class_ids"], r["scores"], r['masks'])
    if not np.isnan(AP):
        APs.append(AP)

print("mAP: ", np.mean(APs))
```

```
mAP:  0.19276911679989675
```

Evaluation method 1b - mean Average Precision (mAP) ignoring classes

In [22]:

```
# Compute VOC-Style mAP @ IoU=0.5
# Running on 100 images. Increase for better accuracy.
image_ids = np.random.choice(dataset_val.image_ids, 150)
#image_ids = dataset_val.image_ids
APs = []
for image_id in image_ids:
    # Load image and ground truth data
    image, image_meta, gt_class_id, gt_bbox, gt_mask =\
        modellib.load_image_gt(dataset_val, inference_config,
                               image_id, use_mini_mask=False)
    molded_images = np.expand_dims(modellib.mold_image(image, inference_config), 0)
    # Run object detection
    results = model.detect([image], verbose=0)
    r = results[0]
    # Ignore classes
    gt_class_id[:] = 1
    r["class_ids"][:] = 1
    # Compute AP
    AP, precisions, recalls, overlaps =\
        utils.compute_ap(gt_bbox, gt_class_id, gt_mask,
                         r["rois"], r["class_ids"], r["scores"], r['masks'])
    if not np.isnan(AP):
        APs.append(AP)

print("mAP (ignoring classes): ", np.mean(APs))
```

```
mAP (ignoring classes):  0.32936668637141897
```

In [23]:

```
r['masks'].shape
#utils.compute_overlaps_masks(gt_mask, r['masks'])
```

Out[23]:

```
(256, 256, 8)
```

Evaluation method 2 - Comparing number of objects detected

In [25]:

```
# Compare number of SExtractor and R-CNN object masks

# number of objects in SExtractor mask:
# len(gt_class_id)
# number of objects in RCNN mask:
# len(r['class_ids'])

detfrac = len(r['class_ids'])/len(gt_class_id)

#SEx
#pd.Series(gt_class_id).value_counts()

#RCNN
#pd.Series(r['class_ids']).value_counts()

print("Fraction of objects detected: ", detfrac)
```

```
Fraction of objects detected:  0.8888888888888888
```

Evaluation method 3 - Visual inspection

In [26]:

```
# Change directory to notes folder
%cd /gpfs01/home/ppzsb1/astobjdet/Mask_RCNN/samples/AstroDetector/Model Notes
```

```
/gpfs01/home/ppzsb1/astobjdet/Mask_RCNN/samples/AstroDetector/Model Notes
```

In [27]:

```
# Add notes below %%writefile line to save notes for visual inspections of resulting images as wel
l as discussion of results
# Change .txt file name each run
```

In [28]:

```
%%writefile m0_s0-5_q8_standardmodel.txt
mAP = 0.1928
mAP (without classes) = 0.3294
Detfrac = 0.888
Visual: Mask_RCNN and SEx highlights the three visual focal points however Mask_RCNN seems to have
trouble distinguishing between the bottom two focals and overestimates the number of objects. SEx p
icks up some smaller galaxies which Mask_RCNN does not.

Discussion: N/A
```

```
Overwriting m0_s0-5_q8_standardmodel.txt
```