Mask RCNN Implementation

Inspect and visualize data loading and pre-processing code.

In [126]:

```
import os
import sys
import itertools
import math
import logging
import json
import re
import random
import time
import concurrent.futures
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.lines as lines
from matplotlib.patches import Polygon
import imgaug
from imgaug import augmenters as iaa

# Root directory of the project
ROOT_DIR = os.getcwd()
if ROOT_DIR.endswith("samples/nucleus"):
    # Go up two levels to the repo root
    ROOT_DIR = os.path.dirname(os.path.dirname(ROOT_DIR))

# Import Mask RCNN
sys.path.append(ROOT_DIR)
from mrcnn import utils
from mrcnn import visualize
from mrcnn.visualize import display_images
from mrcnn import model as modellib
from mrcnn.model import log


import nucleus




%matplotlib inline
```

In [127]:

```
# Comment out to reload imported modules if they change
# %load_ext autoreload
# %autoreload 2
```

Configurations

In [128]:

```
# Dataset directory
DATASET_DIR = os.path.join(ROOT_DIR, "datasets/nucleus")

# Use configuation from nucleus.py, but override
# image resizing so we see the real sizes here
class NoResizeConfig(nucleus.NucleusConfig):
    IMAGE_RESIZE_MODE = "none"

config = NoResizeConfig()
```

Notebook Preferences

In [129]:

```
def get_ax(rows=1, cols=1, size=16):
    """Return a Matplotlib Axes array to be used in
    all visualizations in the notebook. Provide a
    central point to control graph sizes.

    Adjust the size attribute to control how big to render images
    """
    _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
    return ax
```

Dataset

In [130]:

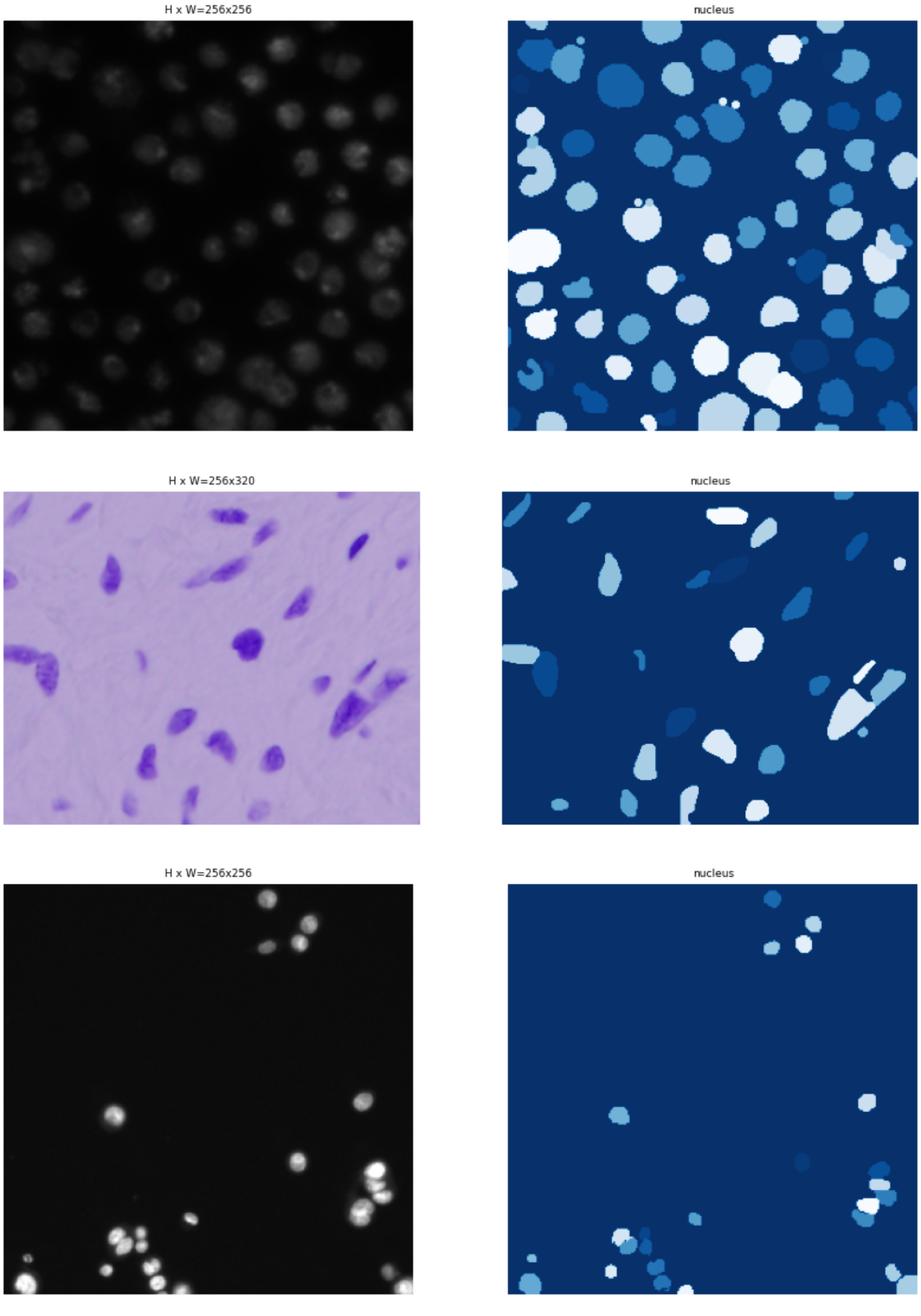
```
# Load dataset
dataset = nucleus.NucleusDataset()
# The subset is the name of the sub-directory, such as stage1_train,
# stage1_test, ...etc. You can also use these special values:
#     train: loads stage1_train but excludes validation images
#     val: loads validation images from stage1_train. For a list
#          of validation images see nucleus.py
dataset.load_nucleus(DATASET_DIR, subset="train")

# Must call before using the dataset
dataset.prepare()

print("Image Count: {}".format(len(dataset.image_ids)))
print("Class Count: {}".format(dataset.num_classes))
for i, info in enumerate(dataset.class_info):
    print("{:3}. {:50}".format(i, info['name']))
```

```
Image Count: 645
Class Count: 2
  0. BG
  1. nucleus
```

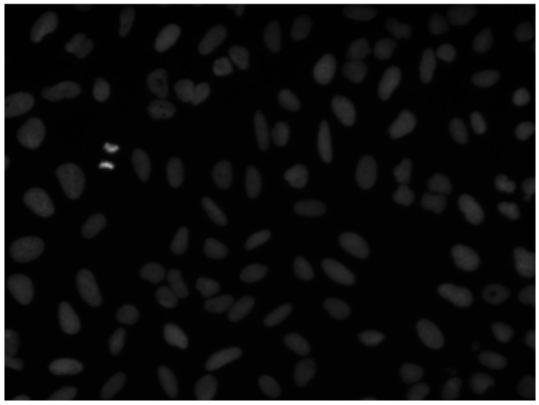
Display Samples

In [131]:

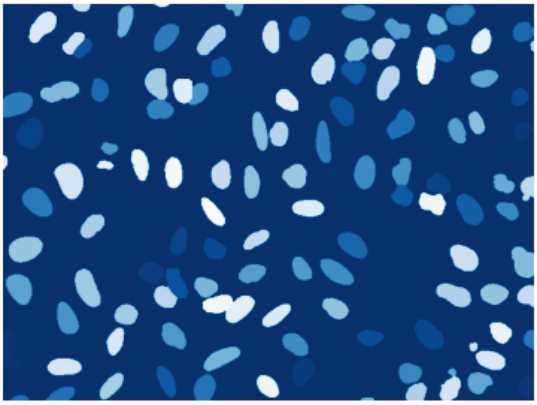
```
# Load and display random samples
image_ids = np.random.choice(dataset.image_ids, 4)
for image_id in image_ids:
    image = dataset.load_image(image_id)
    mask, class_ids = dataset.load_mask(image_id)
    visualize.display_top_masks(image, mask, class_ids, dataset.class_names, limit=1)
```



H x W=520x696



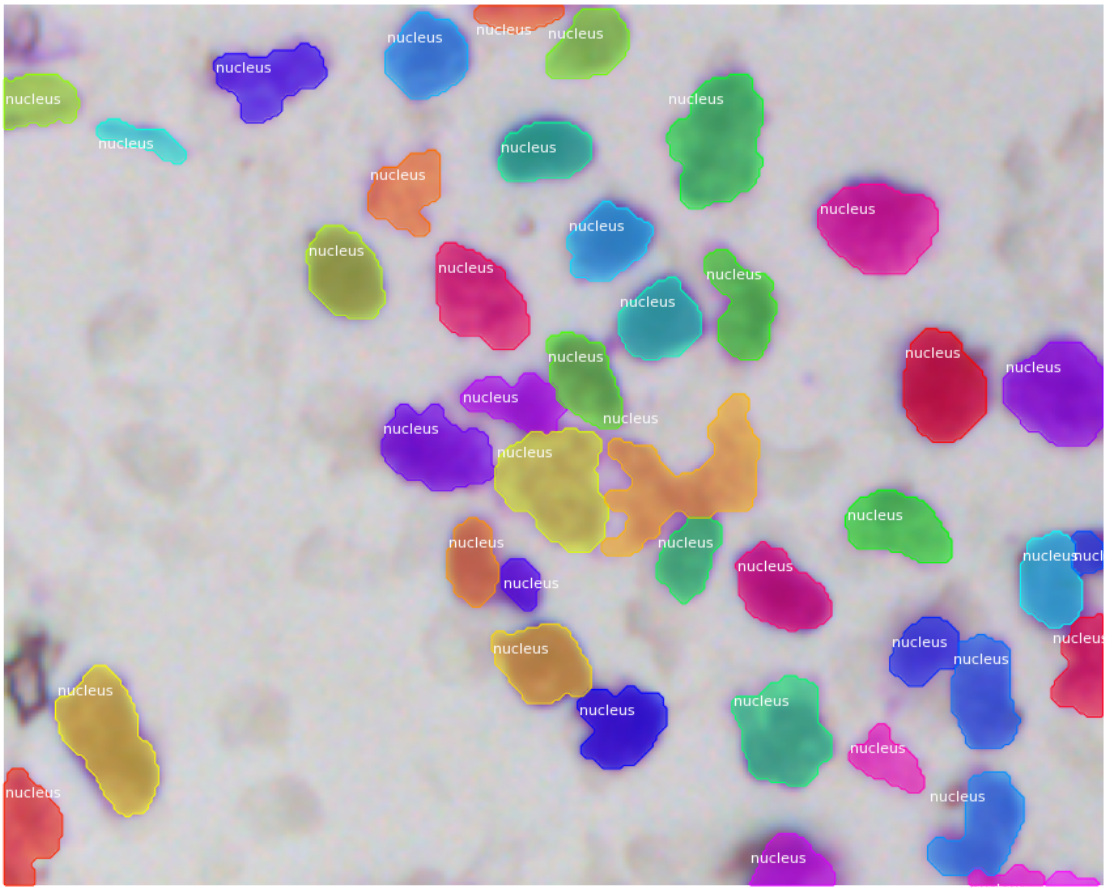
nucleus



In [132]:

```
# Example of loading a specific image by its source ID
source_id = "ed5be4b63e9506ad64660dd92a098ffcc0325195298c13c815a73773f1efc279"

# Map source ID to Dataset image_id
# Notice the nucleus prefix: it's the name given to the dataset in NucleusDataset
image_id = dataset.image_from_source_map["nucleus.{}".format(source_id)]

# Load and display
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
        dataset, config, image_id)
log("molded_image", image)
log("mask", mask)
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names,
                            show_bbox=False)
```

```
molded_image             shape: (256, 320, 3)         min:   28.00000  max:
232.00000  uint8
mask                     shape: (256, 320, 42)        min:    0.00000  max:
1.00000  bool
```



Dataset Stats

Loop through all images in the dataset and collect aggregate stats.

In [133]:

```
def image_stats(image_id):
    """Returns a dict of stats for one image."""
    image = dataset.load_image(image_id)
    mask, _ = dataset.load_mask(image_id)
    bbox = utils.extract_bboxes(mask)
    # Sanity check
    assert mask.shape[:2] == image.shape[:2]
    # Return stats dict
    return {
        "id": image_id,
        "shape": list(image.shape),
        "bbox": [[b[2] - b[0], b[3] - b[1]]
                 for b in bbox
                 # Uncomment to exclude nuclei with 1 pixel width
                 # or height (often on edges)
                 # if b[2] - b[0] > 1 and b[3] - b[1] > 1
                ],
        "color": np.mean(image, axis=(0, 1)),
    }

# Loop through the dataset and compute stats over multiple threads
# This might take a few minutes
t_start = time.time()
with concurrent.futures.ThreadPoolExecutor() as e:
    stats = list(e.map(image_stats, dataset.image_ids))
t_total = time.time() - t_start
print("Total time: {:.1f} seconds".format(t_total))
```

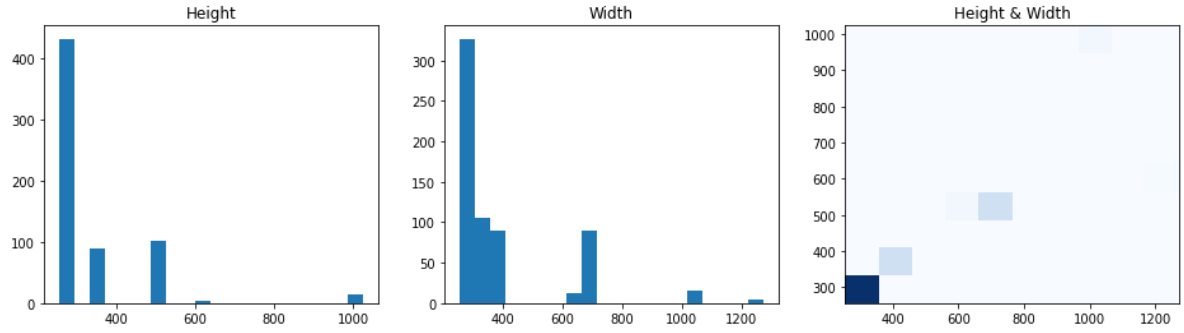
```
Total time: 108.3 seconds
```

Image Size Stats

In [134]:

```
# Image stats
image_shape = np.array([s['shape'] for s in stats])
image_color = np.array([s['color'] for s in stats])
print("Image Count: ", image_shape.shape[0])
print("Height  mean: {:.2f}  median: {:.2f}  min: {:.2f}  max: {:.2f}".format(
    np.mean(image_shape[:, 0]), np.median(image_shape[:, 0]),
    np.min(image_shape[:, 0]), np.max(image_shape[:, 0])))
print("Width   mean: {:.2f}  median: {:.2f}  min: {:.2f}  max: {:.2f}".format(
    np.mean(image_shape[:, 1]), np.median(image_shape[:, 1]),
    np.min(image_shape[:, 1]), np.max(image_shape[:, 1])))
print("Color   mean (RGB): {:.2f} {:.2f} {:.2f}".format(*np.mean(image_color, axis=0)))

# Histograms
fig, ax = plt.subplots(1, 3, figsize=(16, 4))
ax[0].set_title("Height")
_ = ax[0].hist(image_shape[:, 0], bins=20)
ax[1].set_title("Width")
_ = ax[1].hist(image_shape[:, 1], bins=20)
ax[2].set_title("Height & Width")
_ = ax[2].hist2d(image_shape[:, 1], image_shape[:, 0], bins=10, cmap="Blues")
```

```
Image Count:  645
Height  mean: 332.55  median: 256.00  min: 256.00  max: 1024.00
Width   mean: 374.53  median: 256.00  min: 256.00  max: 1272.00
Color   mean (RGB): 41.70 37.98 46.30
```

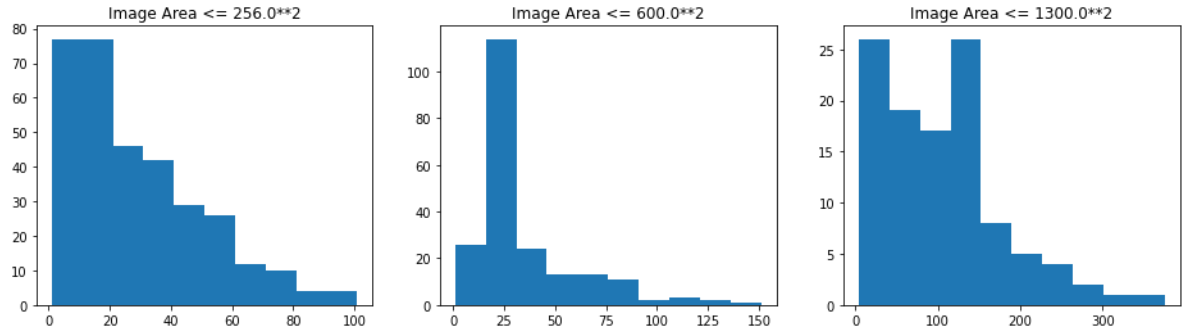


Nuclei per Image Stats

In [135]:

```
# Segment by image area
image_area_bins = [256**2, 600**2, 1300**2]

print("Nuclei/Image")
fig, ax = plt.subplots(1, len(image_area_bins), figsize=(16, 4))
area_threshold = 0
for i, image_area in enumerate(image_area_bins):
    nuclei_per_image = np.array([len(s['bbox'])
                                 for s in stats
                                 if area_threshold < (s['shape'][0] * s['shape'][1]) <= ima
    area_threshold = image_area
    if len(nuclei_per_image) == 0:
        print("Image area <= {:4}**2: None".format(np.sqrt(image_area)))
        continue
    print("Image area <= {:4.0f}**2:  mean: {:.1f}  median: {:.1f}  min: {:.1f}  max: {:.1f
        np.sqrt(image_area), nuclei_per_image.mean(), np.median(nuclei_per_image),
        nuclei_per_image.min(), nuclei_per_image.max()))
    ax[i].set_title("Image Area <= {:4}**2".format(np.sqrt(image_area)))
    _ = ax[i].hist(nuclei_per_image, bins=10)
```

```
Nuclei/Image
Image area <=  256**2:  mean: 28.7  median: 22.0  min: 1.0  max: 101.0
Image area <=  600**2:  mean: 34.3  median: 25.0  min: 1.0  max: 151.0
Image area <= 1300**2:  mean: 104.6  median: 101.0  min: 4.0  max: 375.0
```

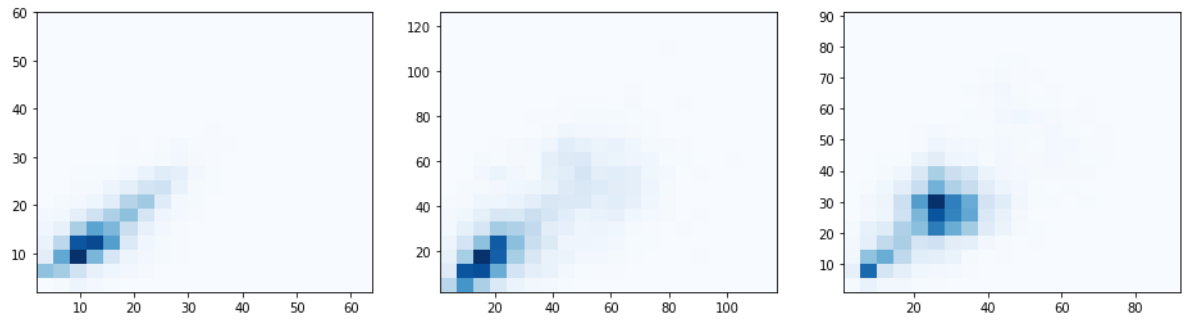


Nuclei Size Stats

In [136]:

```
# Nuclei size stats
fig, ax = plt.subplots(1, len(image_area_bins), figsize=(16, 4))
area_threshold = 0
for i, image_area in enumerate(image_area_bins):
    nucleus_shape = np.array([
        b
        for s in stats if area_threshold < (s['shape'][0] * s['shape'][1]) <= image_area
        for b in s['bbox']])
    nucleus_area = nucleus_shape[:, 0] * nucleus_shape[:, 1]
    area_threshold = image_area

    print("\nImage Area <= {:.0f}**2".format(np.sqrt(image_area)))
    print("  Total Nuclei: ", nucleus_shape.shape[0])
    print("  Nucleus Height. mean: {:.2f}  median: {:.2f}  min: {:.2f}  max: {:.2f}".format
        np.mean(nucleus_shape[:, 0]), np.median(nucleus_shape[:, 0]),
        np.min(nucleus_shape[:, 0]), np.max(nucleus_shape[:, 0])))
    print("  Nucleus Width.  mean: {:.2f}  median: {:.2f}  min: {:.2f}  max: {:.2f}".format
        np.mean(nucleus_shape[:, 1]), np.median(nucleus_shape[:, 1]),
        np.min(nucleus_shape[:, 1]), np.max(nucleus_shape[:, 1])))
    print("  Nucleus Area.   mean: {:.2f}  median: {:.2f}  min: {:.2f}  max: {:.2f}".format
        np.mean(nucleus_area), np.median(nucleus_area),
        np.min(nucleus_area), np.max(nucleus_area)))

    # Show 2D histogram
    _ = ax[i].hist2d(nucleus_shape[:, 1], nucleus_shape[:, 0], bins=20, cmap="Blues")
```

```
Image Area <= 256**2
  Total Nuclei:  9378
  Nucleus Height. mean: 14.14  median: 13.00  min: 2.00  max: 60.00
  Nucleus Width.  mean: 14.38  median: 13.00  min: 2.00  max: 64.00
  Nucleus Area.   mean: 232.19  median: 168.00  min: 24.00  max: 2880.00

Image Area <= 600**2
  Total Nuclei:  7177
  Nucleus Height. mean: 28.53  median: 23.00  min: 2.00  max: 126.00
  Nucleus Width.  mean: 29.56  median: 24.00  min: 1.00  max: 117.00
  Nucleus Area.   mean: 1086.77  median: 529.00  min: 21.00  max: 13560.00

Image Area <= 1300**2
  Total Nuclei:  11402
  Nucleus Height. mean: 25.82  median: 26.00  min: 1.00  max: 91.00
  Nucleus Width.  mean: 25.88  median: 26.00  min: 1.00  max: 92.00
  Nucleus Area.   mean: 747.15  median: 688.00  min: 21.00  max: 5520.00
```



In [137]:

```
# Nuclei height/width ratio
nucleus_aspect_ratio = nucleus_shape[:, 0] / nucleus_shape[:, 1]
print("Nucleus Aspect Ratio.  mean: {:.2f}  median: {:.2f}  min: {:.2f}  max: {:.2f}".forma
    np.mean(nucleus_aspect_ratio), np.median(nucleus_aspect_ratio),
    np.min(nucleus_aspect_ratio), np.max(nucleus_aspect_ratio)))
plt.figure(figsize=(15, 5))
_ = plt.hist(nucleus_aspect_ratio, bins=100, range=[0, 5])
```

```
Nucleus Aspect Ratio.  mean: 1.09  median: 1.00  min: 0.02  max: 57.00
```

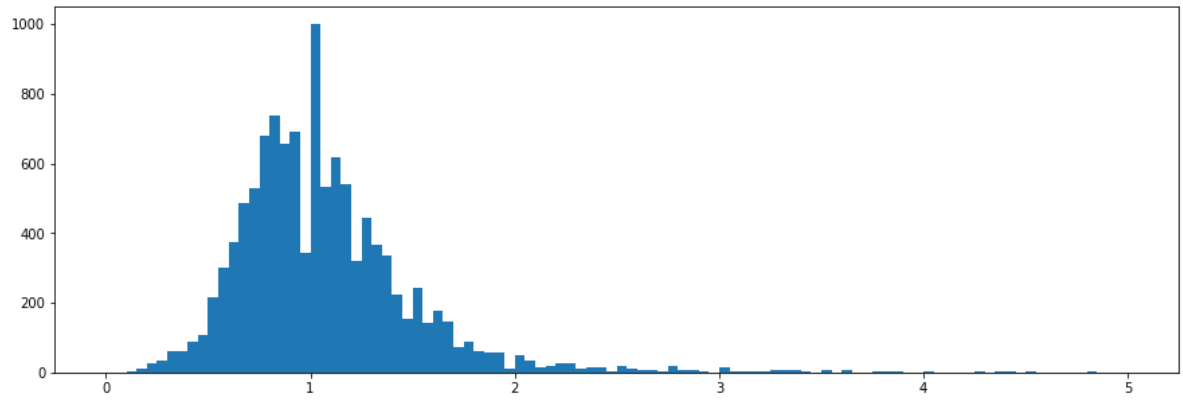


Image Augmentation

Test out different augmentation methods

In [138]:

```
# List of augmentations
# http://imgaug.readthedocs.io/en/latest/source/augmenters.html
augmentation = iaa.Sometimes(0.9, [
    iaa.Fliplr(0.5),
    iaa.Flipud(0.5),
    iaa.Multiply((0.8, 1.2)),
    iaa.GaussianBlur(sigma=(0.0, 5.0))
])
```

In [139]:

```
# Load the image multiple times to show augmentations
limit = 4
ax = get_ax(rows=2, cols=limit//2)
for i in range(limit):
    image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
        dataset, config, image_id, augment=False, augmentation=augmentation)
    visualize.display_instances(image, bbox, mask, class_ids,
                                dataset.class_names, ax=ax[i//2, i % 2],
                                show_mask=False, show_bbox=False)
```

```
C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\site-pack
ages\imgaug\augmenters\base.py:49: SuspiciousSingleImageShapeWarning: You pr
ovided a numpy array of shape (256, 320, 42) as a single-image augmentation
input, which was interpreted as (H, W, C). The last dimension however has a
size of >=32, which indicates that you provided a multi-image array with sha
pe (N, H, W) instead. If that is the case, you should use e.g. augmenter(ima
geS=<your input>) or augment_imageS(<your input>). Otherwise your multi-imag
e input will be interpreted as a single image during augmentation.
  ia.warn(
C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\site-pack
ages\imgaug\augmenters\base.py:49: SuspiciousSingleImageShapeWarning: You pr
ovided a numpy array of shape (256, 320, 42) as a single-image augmentation
input, which was interpreted as (H, W, C). The last dimension however has a
size of >=32, which indicates that you provided a multi-image array with sha
pe (N, H, W) instead. If that is the case, you should use e.g. augmenter(ima
geS=<your input>) or augment_imageS(<your input>). Otherwise your multi-imag
e input will be interpreted as a single image during augmentation.
  ia.warn(
C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\site-pack
ages\imgaug\augmenters\base.py:49: SuspiciousSingleImageShapeWarning: You pr
ovided a numpy array of shape (256, 320, 42) as a single-image augmentation
input, which was interpreted as (H, W, C). The last dimension however has a
size of >=32, which indicates that you provided a multi-image array with sha
pe (N, H, W) instead. If that is the case, you should use e.g. augmenter(ima
geS=<your input>) or augment_imageS(<your input>). Otherwise your multi-imag
e input will be interpreted as a single image during augmentation.
  ia.warn(
C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\site-pack
ages\imgaug\augmenters\base.py:49: SuspiciousSingleImageShapeWarning: You pr
ovided a numpy array of shape (256, 320, 42) as a single-image augmentation
input, which was interpreted as (H, W, C). The last dimension however has a
size of >=32, which indicates that you provided a multi-image array with sha
pe (N, H, W) instead. If that is the case, you should use e.g. augmenter(ima
geS=<your input>) or augment_imageS(<your input>). Otherwise your multi-imag
e input will be interpreted as a single image during augmentation.
  ia.warn(
```

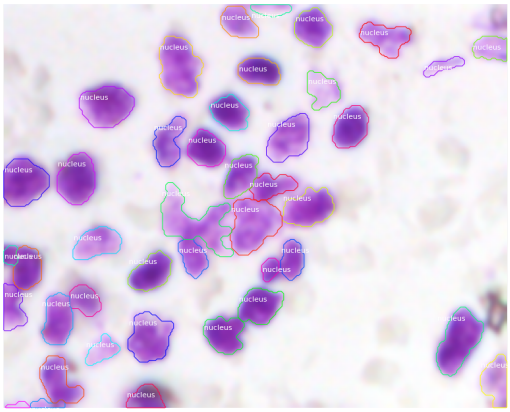
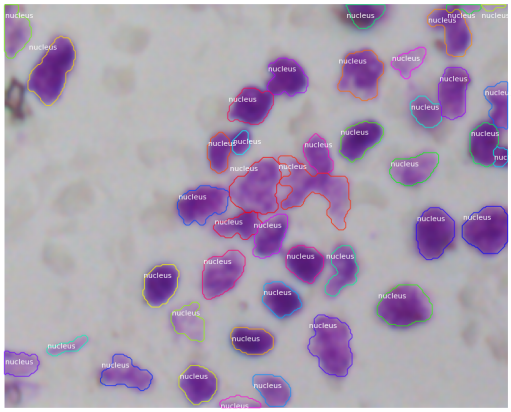
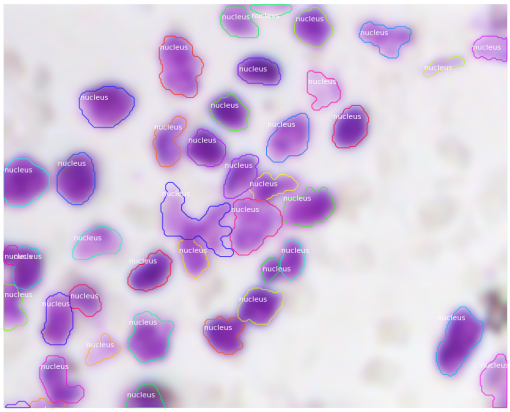
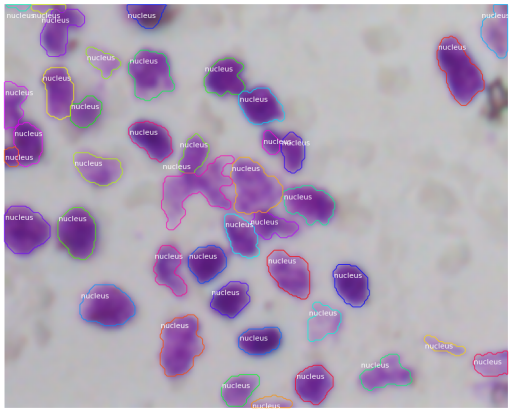





Image Crops

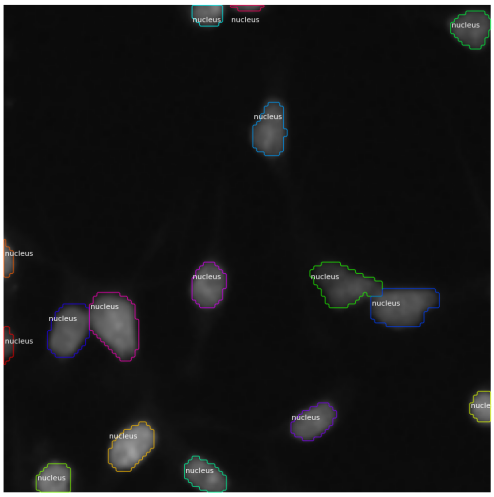
Microscoy images tend to be large, but nuclei are small. So it's more efficient to train on random crops from large images. This is handled by `config.IMAGE_RESIZE_MODE = "crop"` .

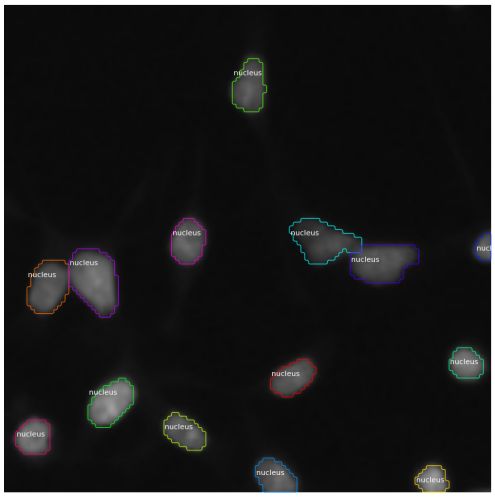
In [140]:

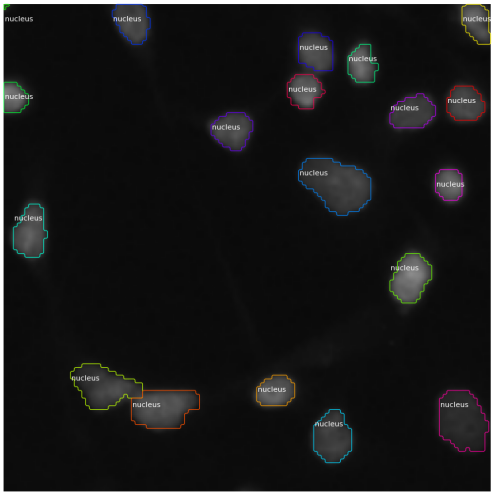
```
class RandomCropConfig(nucleus.NucleusConfig):
    IMAGE_RESIZE_MODE = "crop"
    IMAGE_MIN_DIM = 256
    IMAGE_MAX_DIM = 256

crop_config = RandomCropConfig()
```

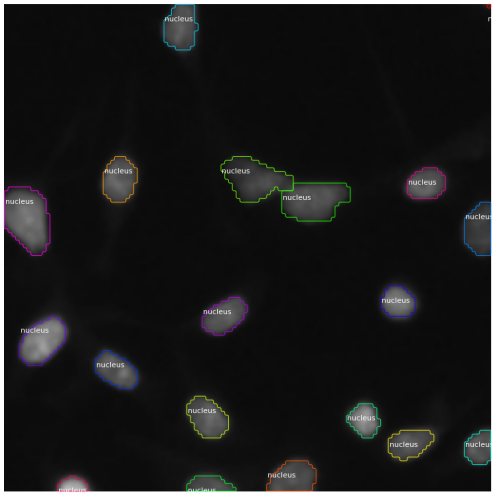
In [141]:

```
# Load the image multiple times to show augmentations
limit = 4
image_id = np.random.choice(dataset.image_ids, 1)[0]
ax = get_ax(rows=2, cols=limit//2)
for i in range(limit):
    image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
        dataset, crop_config, image_id)
    visualize.display_instances(image, bbox, mask, class_ids,
                                dataset.class_names, ax=ax[i//2, i % 2],
                                show_mask=False, show_bbox=False)
```









Mini Masks

Instance binary masks can get large when training with high resolution images. For example, if training with 1024x1024 image then the mask of a single instance requires 1MB of memory (Numpy uses bytes for boolean values). If an image has 100 instances then that's 100MB for the masks alone.

To improve training speed, we optimize masks:

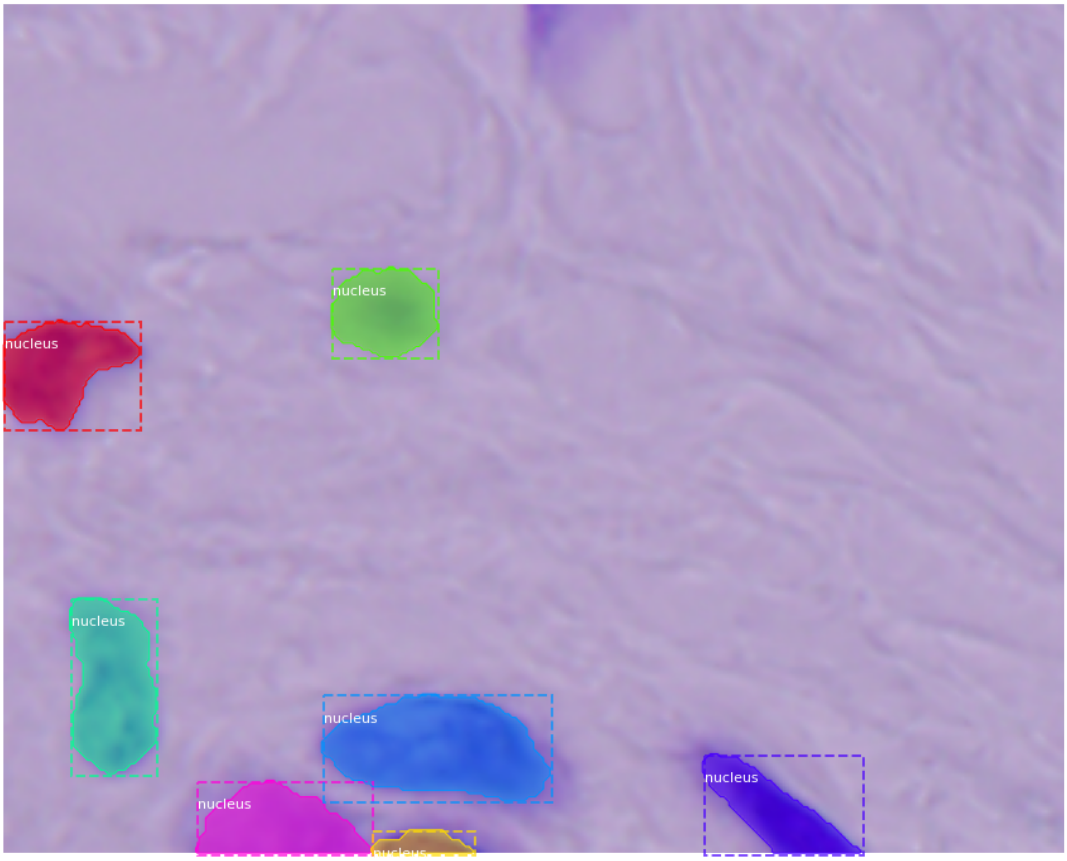
- We store mask pixels that are inside the object bounding box, rather than a mask of the full image. Most objects are small compared to the image size, so we save space by not storing a lot of zeros around the object.
- We resize the mask to a smaller size (e.g. 56x56). For objects that are larger than the selected size we lose a bit of accuracy. But most object annotations are not very accuracy to begin with, so this loss is negligable for most practical purposes. Thie size of the mini_mask can be set in the config class.

To visualize the effect of mask resizing, and to verify the code correctness, we visualize some examples.

In [142]:

```
# Load random image and mask.
image_id = np.random.choice(dataset.image_ids, 1)[0]
image = dataset.load_image(image_id)
mask, class_ids = dataset.load_mask(image_id)
original_shape = image.shape
# Resize
image, window, scale, padding, _ = utils.resize_image(
    image,
    min_dim=config.IMAGE_MIN_DIM,
    max_dim=config.IMAGE_MAX_DIM,
    mode=config.IMAGE_RESIZE_MODE)
mask = utils.resize_mask(mask, scale, padding)
# Compute Bounding box
bbox = utils.extract_bboxes(mask)

# Display image and additional stats
print("image_id: ", image_id, dataset.image_reference(image_id))
print("Original shape: ", original_shape)
log("image", image)
log("mask", mask)
log("class_ids", class_ids)
log("bbox", bbox)
# Display image and instances
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

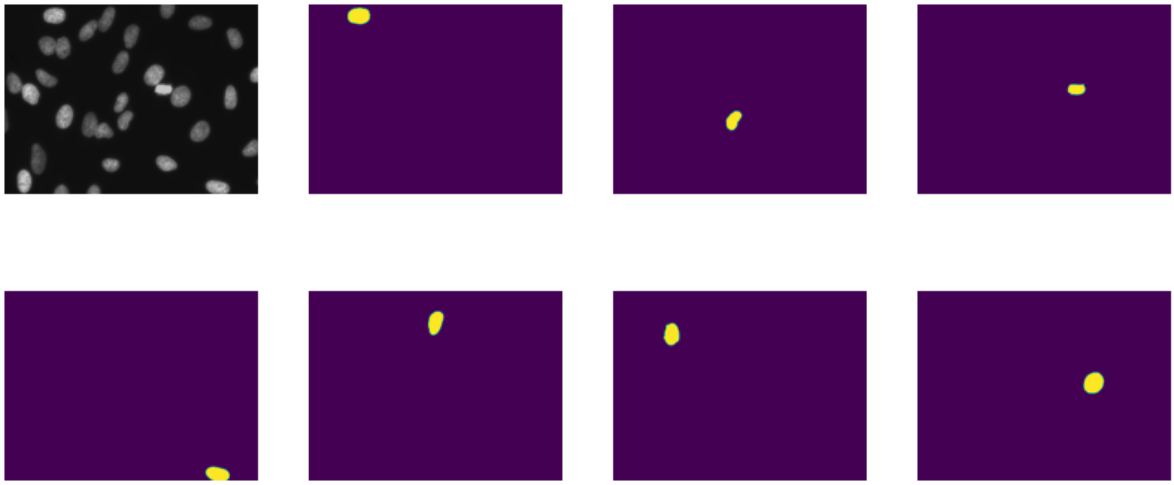
```
image_id:  422 01d44a26f6680c42ba94c9bc6339228579a95d0e2695b149b7cc0c9592b21
baf
Original shape:  (256, 320, 3)
image                    shape: (256, 320, 3)         min:    1.00000  max:
217.00000  uint8
mask                     shape: (256, 320, 7)         min:    0.00000  max:
1.00000  bool
class_ids                shape: (7,)                  min:    1.00000  max:
1.00000  int32
bbox                     shape: (7, 4)                min:    0.00000  max:
259.00000  int32
```


nucleus
nucleus
nucleus
nucleus
nucleus
nucleus

In [143]:

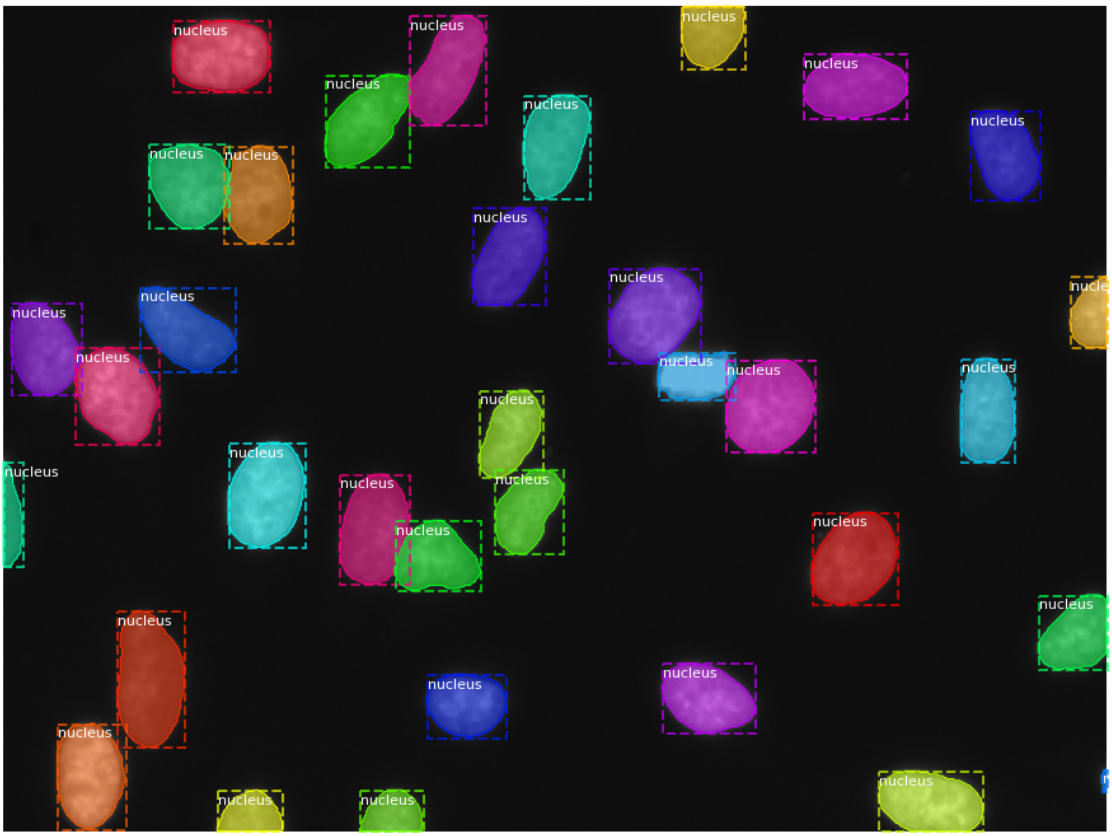
```
image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
    dataset, config, image_id)

log("image", image)
log("image_meta", image_meta)
log("class_ids", class_ids)
log("bbox", bbox)
log("mask", mask)

display_images([image]+[mask[:,:,i] for i in range(min(mask.shape[-1], 7))])
```

```
image                    shape: (520, 696, 3)         min:   11.00000  max:
226.00000  uint8
image_meta               shape: (14,)                 min:    0.00000  max:
696.00000  int32
class_ids                shape: (34,)                 min:    1.00000  max:
1.00000  int32
bbox                     shape: (34, 4)               min:    0.00000  max:
696.00000  int32
mask                     shape: (520, 696, 34)        min:    0.00000  max:
1.00000  bool
```



In [144]:

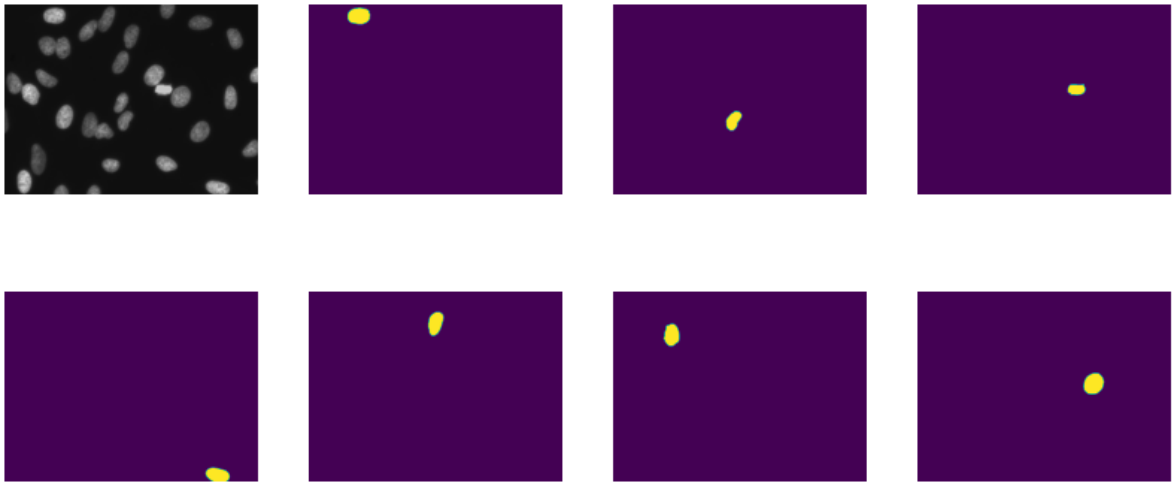
```
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```



In [145]:

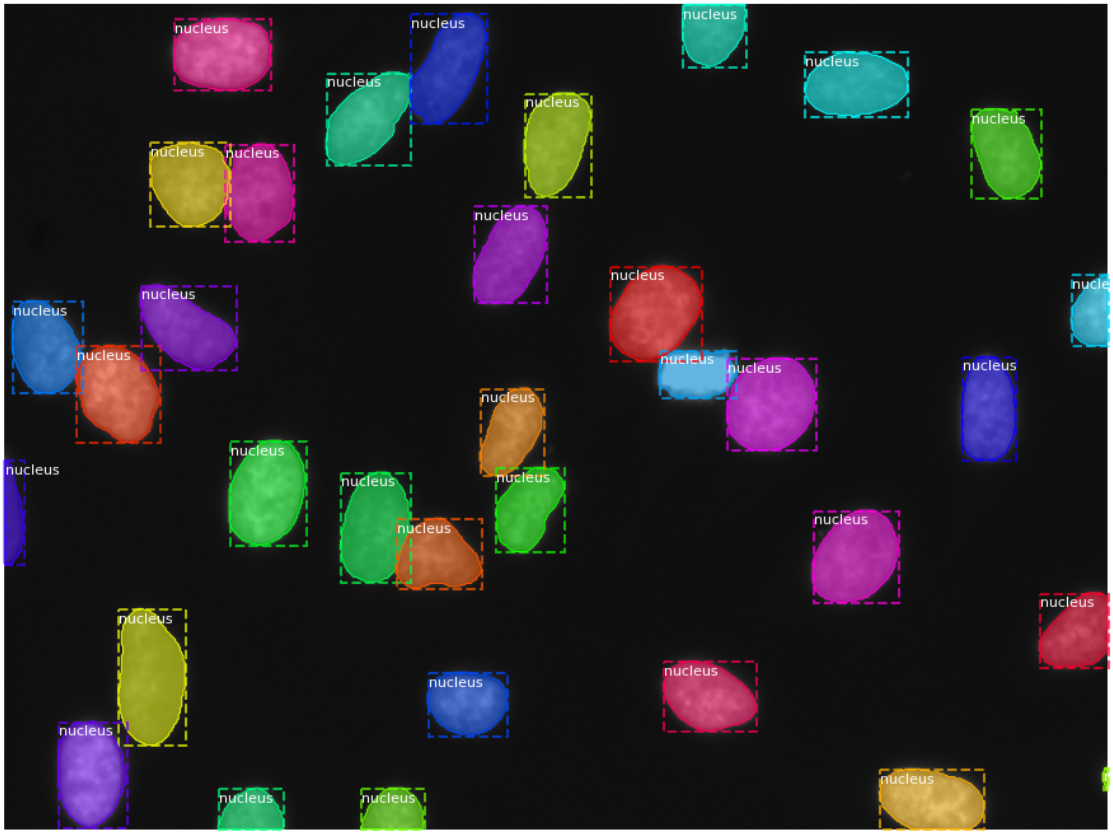
```
# Add augmentation and mask resizing.
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(dataset, config, image_id
log("mask", mask)
display_images([image]+[mask[:,:,i] for i in range(min(mask.shape[-1], 7))])
```

```
WARNING:root:'augment' is depricated. Use 'augmentation' instead.

mask                     shape: (520, 696, 34)        min:    0.00000  max:
1.00000  bool
```



In [146]:

```
# mask = utils.expand_mask(bbox, mask, image.shape)
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```



Anchors

For an FPN network, the anchors must be ordered in a way that makes it easy to match anchors to the output of the convolution layers that predict anchor scores and shifts.

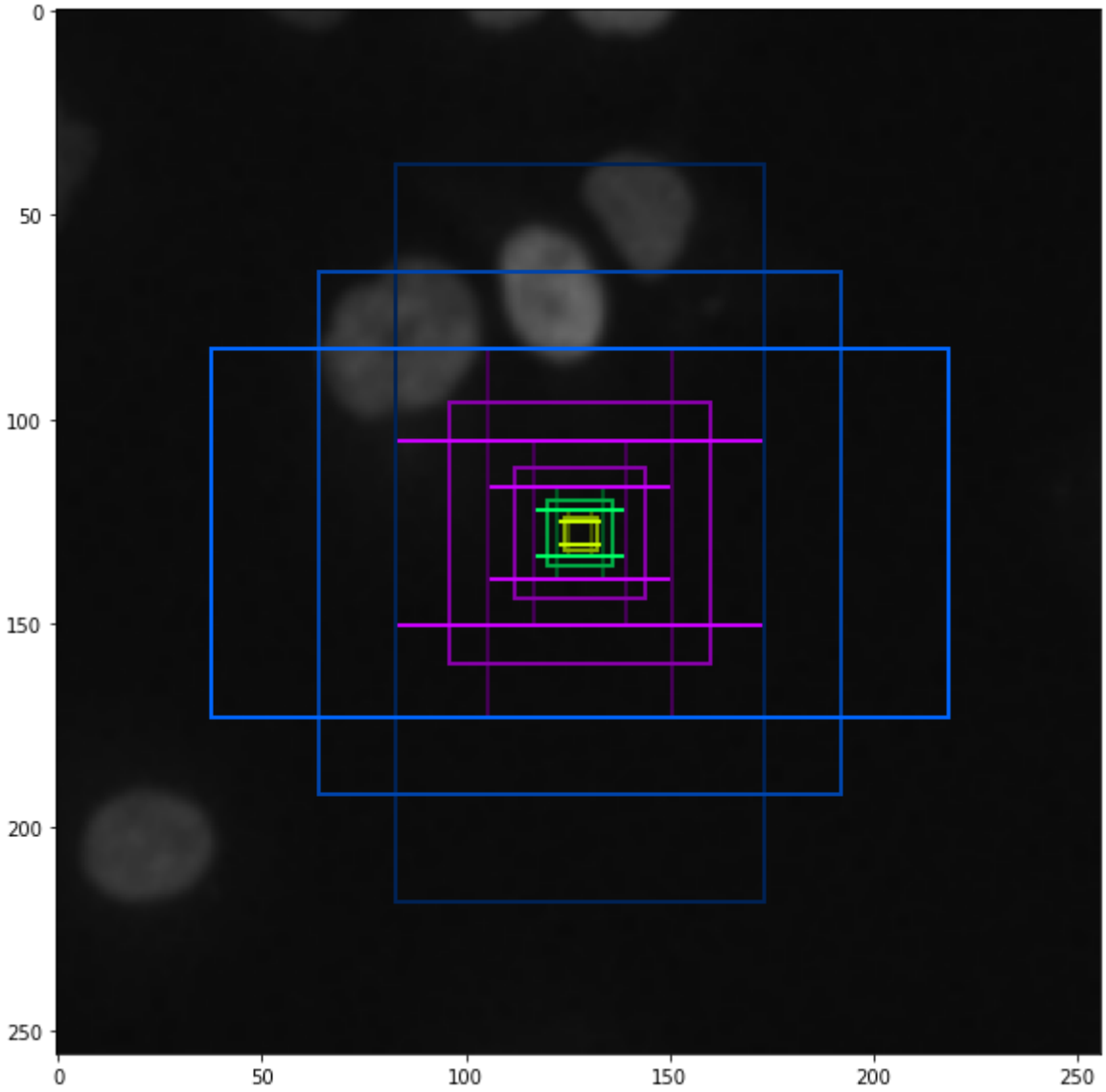
- Sort by pyramid level first. All anchors of the first level, then all of the second and so on. This makes it easier to separate anchors by level.
- Within each level, sort anchors by feature map processing sequence. Typically, a convolution layer processes a feature map starting from top-left and moving right row by row.
- For each feature map cell, pick any sorting order for the anchors of different ratios. Here we match the order of ratios passed to the function.

In [147]:

```
## Visualize anchors of one cell at the center of the feature map

# Load and display random image
image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, _, _, _ = modellib.load_image_gt(dataset, crop_config, image_id)

# Generate Anchors
backbone_shapes = modellib.compute_backbone_shapes(config, image.shape)
anchors = utils.generate_pyramid_anchors(config.RPN_ANCHOR_SCALES,
                                         config.RPN_ANCHOR_RATIOS,
                                         backbone_shapes,
                                         config.BACKBONE_STRIDES,
                                         config.RPN_ANCHOR_STRIDE)

# Print summary of anchors
num_levels = len(backbone_shapes)
anchors_per_cell = len(config.RPN_ANCHOR_RATIOS)
print("Count: ", anchors.shape[0])
print("Scales: ", config.RPN_ANCHOR_SCALES)
print("ratios: ", config.RPN_ANCHOR_RATIOS)
print("Anchors per Cell: ", anchors_per_cell)
print("Levels: ", num_levels)
anchors_per_level = []
for l in range(num_levels):
    num_cells = backbone_shapes[l][0] * backbone_shapes[l][1]
    anchors_per_level.append(anchors_per_cell * num_cells // config.RPN_ANCHOR_STRIDE**2)
    print("Anchors in Level {}: {}".format(l, anchors_per_level[l]))

# Display
fig, ax = plt.subplots(1, figsize=(10, 10))
ax.imshow(image)
levels = len(backbone_shapes)

for level in range(levels):
    colors = visualize.random_colors(levels)
    # Compute the index of the anchors at the center of the image
    level_start = sum(anchors_per_level[:level]) # sum of anchors of previous levels
    level_anchors = anchors[level_start:level_start+anchors_per_level[level]]
    print("Level {}. Anchors: {:6}  Feature map Shape: {}".format(level, level_anchors.shap
                                                                  backbone_shapes[level]))
    center_cell = backbone_shapes[level] // 2
    center_cell_index = (center_cell[0] * backbone_shapes[level][1] + center_cell[1])
    level_center = center_cell_index * anchors_per_cell
    center_anchor = anchors_per_cell * (
        (center_cell[0] * backbone_shapes[level][1] / config.RPN_ANCHOR_STRIDE**2) \
        + center_cell[1] / config.RPN_ANCHOR_STRIDE)
    level_center = int(center_anchor)

    # Draw anchors. Brightness show the order in the array, dark to bright.
    for i, rect in enumerate(level_anchors[level_center:level_center+anchors_per_cell]):
        y1, x1, y2, x2 = rect
        p = patches.Rectangle((x1, y1), x2-x1, y2-y1, linewidth=2, facecolor='none',
                              edgecolor=(i+1)*np.array(colors[level]) / anchors_per_cell)
        ax.add_patch(p)
```

```
Count:  16368
Scales:  (8, 16, 32, 64, 128)
```

```
ratios:  [0.5, 1, 2]
Anchors per Cell:  3
Levels:  5
Anchors in Level 0: 12288
Anchors in Level 1: 3072
Anchors in Level 2: 768
Anchors in Level 3: 192
Anchors in Level 4: 48
Level 0. Anchors:  12288  Feature map Shape: [64 64]
Level 1. Anchors:   3072  Feature map Shape: [32 32]
Level 2. Anchors:    768  Feature map Shape: [16 16]
Level 3. Anchors:    192  Feature map Shape: [8 8]
Level 4. Anchors:     48  Feature map Shape: [4 4]
```



Data Generator

In [148]:

```
# Create data generator
random_rois = 2000
g = modellib.data_generator(
    dataset, crop_config, shuffle=True, random_rois=random_rois,
    batch_size=4,
    detection_targets=True)
```

In [149]:

```
# Uncomment to run the generator through a lot of images
# to catch rare errors
# for i in range(1000):
#     print(i)
#     _, _ = next(g)
```

In [150]:

```
# Get Next Image
if random_rois:
    [normalized_images, image_meta, rpn_match, rpn_bbox, gt_class_ids, gt_boxes, gt_masks,
    [mrcnn_class_ids, mrcnn_bbox, mrcnn_mask] = next(g)

    log("rois", rois)
    log("mrcnn_class_ids", mrcnn_class_ids)
    log("mrcnn_bbox", mrcnn_bbox)
    log("mrcnn_mask", mrcnn_mask)
else:
    [normalized_images, image_meta, rpn_match, rpn_bbox, gt_boxes, gt_masks], _ = next(g)

log("gt_class_ids", gt_class_ids)
log("gt_boxes", gt_boxes)
log("gt_masks", gt_masks)
log("rpn_match", rpn_match, )
log("rpn_bbox", rpn_bbox)
image_id = modellib.parse_image_meta(image_meta)["image_id"][0]
print("image_id: ", image_id, dataset.image_reference(image_id))

# Remove the last dim in mrcnn_class_ids. It's only added
# to satisfy Keras restriction on target shape.
mrcnn_class_ids = mrcnn_class_ids[:,:,0]
```

```
ERROR:root:Error processing image {'id': 'a6593632dcbbe4c9e9429a9cec573d26fd
8c91a47d554d315f25e7c2e0280ee3', 'source': 'nucleus', 'path': 'C:\\Users\\Ha
riket Sheth\\Desktop\\MaskRCNN-nuclei-segmentation-using-tensorflow-master
\\MaskRCNN-nuclei-segmentation-using-tensorflow-master\\nucleus\\datasets/nu
cleus\\stage1_train\\a6593632dcbbe4c9e9429a9cec573d26fd8c91a47d554d315f25e7c
2e0280ee3\\images/a6593632dcbbe4c9e9429a9cec573d26fd8c91a47d554d315f25e7c2e0
280ee3.png'}
Traceback (most recent call last):
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1702, in data_generator
    load_image_gt(dataset, config, image_id, augment=augment,
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1287, in load_image_gt
    mask = utils.minimize_mask(bbox, mask, config.MINI_MASK_SHAPE)
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\utils.py", line 541, in minimize_mask
    m = skimage.transform.resize(m, mini_shape, order=1, mode="constant")
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\transform\_warps.py", line 160, in resize
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\_shared\utils.py", line 724, in _validate_interpolation
_order
ValueError: Input image dtype is bool. Interpolation is not defined with boo
l data type. Please set order to 0 or explicitely cast input image to anothe
r data type.
ERROR:root:Error processing image {'id': '718751b439c05bdd589f04fcef321a86be
3ecb35292a435138e295e05eb2e771', 'source': 'nucleus', 'path': 'C:\\Users\\Ha
riket Sheth\\Desktop\\MaskRCNN-nuclei-segmentation-using-tensorflow-master
\\MaskRCNN-nuclei-segmentation-using-tensorflow-master\\nucleus\\datasets/nu
cleus\\stage1_train\\718751b439c05bdd589f04fcef321a86be3ecb35292a435138e295e
05eb2e771\\images/718751b439c05bdd589f04fcef321a86be3ecb35292a435138e295e05e
b2e771.png'}
Traceback (most recent call last):
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1702, in data_generator
```

```
    load_image_gt(dataset, config, image_id, augment=augment,
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1287, in load_image_gt
    mask = utils.minimize_mask(bbox, mask, config.MINI_MASK_SHAPE)
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\utils.py", line 541, in minimize_mask
    m = skimage.transform.resize(m, mini_shape, order=1, mode="constant")
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\transform\_warps.py", line 160, in resize
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\_shared\utils.py", line 724, in _validate_interpolation
_order
ValueError: Input image dtype is bool. Interpolation is not defined with boo
l data type. Please set order to 0 or explicitely cast input image to anothe
r data type.
ERROR:root:Error processing image {'id': '77ceeb87f560775ac150b8b9b09684ed3e
806d0af6f26cce8f10c5fc280f5df2', 'source': 'nucleus', 'path': 'C:\\Users\\Ha
riket Sheth\\Desktop\\MaskRCNN-nuclei-segmentation-using-tensorflow-master
\\MaskRCNN-nuclei-segmentation-using-tensorflow-master\\nucleus\\datasets/nu
cleus\\stage1_train\\77ceeb87f560775ac150b8b9b09684ed3e806d0af6f26cce8f10c5f
c280f5df2\\images/77ceeb87f560775ac150b8b9b09684ed3e806d0af6f26cce8f10c5fc28
0f5df2.png'}
Traceback (most recent call last):
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1702, in data_generator
    load_image_gt(dataset, config, image_id, augment=augment,
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1287, in load_image_gt
    mask = utils.minimize_mask(bbox, mask, config.MINI_MASK_SHAPE)
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\utils.py", line 541, in minimize_mask
    m = skimage.transform.resize(m, mini_shape, order=1, mode="constant")
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\transform\_warps.py", line 160, in resize
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\_shared\utils.py", line 724, in _validate_interpolation
_order
ValueError: Input image dtype is bool. Interpolation is not defined with boo
l data type. Please set order to 0 or explicitely cast input image to anothe
r data type.
ERROR:root:Error processing image {'id': '8de0b1a2e8f614af29fe5fafeaa5bdf55e
6b3e65edf36355f19b707f7649ce2b', 'source': 'nucleus', 'path': 'C:\\Users\\Ha
riket Sheth\\Desktop\\MaskRCNN-nuclei-segmentation-using-tensorflow-master
\\MaskRCNN-nuclei-segmentation-using-tensorflow-master\\nucleus\\datasets/nu
cleus\\stage1_train\\8de0b1a2e8f614af29fe5fafeaa5bdf55e6b3e65edf36355f19b707
f7649ce2b\\images/8de0b1a2e8f614af29fe5fafeaa5bdf55e6b3e65edf36355f19b707f76
49ce2b.png'}
Traceback (most recent call last):
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1702, in data_generator
    load_image_gt(dataset, config, image_id, augment=augment,
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1287, in load_image_gt
    mask = utils.minimize_mask(bbox, mask, config.MINI_MASK_SHAPE)
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\utils.py", line 541, in minimize_mask
    m = skimage.transform.resize(m, mini_shape, order=1, mode="constant")
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\transform\_warps.py", line 160, in resize
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\_shared\utils.py", line 724, in _validate_interpolation
```

```
_order
ValueError: Input image dtype is bool. Interpolation is not defined with boo
l data type. Please set order to 0 or explicitely cast input image to anothe
r data type.
ERROR:root:Error processing image {'id': '1815cf307859b3e13669041d181aa3b3db
bac1a95aef4c42164b223110c09168', 'source': 'nucleus', 'path': 'C:\\Users\\Ha
riket Sheth\\Desktop\\MaskRCNN-nuclei-segmentation-using-tensorflow-master
\\MaskRCNN-nuclei-segmentation-using-tensorflow-master\\nucleus\\datasets/nu
cleus\\stage1_train\\1815cf307859b3e13669041d181aa3b3dbbac1a95aef4c42164b223
110c09168\\images/1815cf307859b3e13669041d181aa3b3dbbac1a95aef4c42164b223110
c09168.png'}
Traceback (most recent call last):
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1702, in data_generator
    load_image_gt(dataset, config, image_id, augment=augment,
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1287, in load_image_gt
    mask = utils.minimize_mask(bbox, mask, config.MINI_MASK_SHAPE)
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\utils.py", line 541, in minimize_mask
    m = skimage.transform.resize(m, mini_shape, order=1, mode="constant")
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\transform\_warps.py", line 160, in resize
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\_shared\utils.py", line 724, in _validate_interpolation
_order
ValueError: Input image dtype is bool. Interpolation is not defined with boo
l data type. Please set order to 0 or explicitely cast input image to anothe
r data type.
ERROR:root:Error processing image {'id': '8e8a7a14749d0b2e48de3d10e2e80063f1
7b165ad921c8afc0623f08500f3259', 'source': 'nucleus', 'path': 'C:\\Users\\Ha
riket Sheth\\Desktop\\MaskRCNN-nuclei-segmentation-using-tensorflow-master
\\MaskRCNN-nuclei-segmentation-using-tensorflow-master\\nucleus\\datasets/nu
cleus\\stage1_train\\8e8a7a14749d0b2e48de3d10e2e80063f17b165ad921c8afc0623f0
8500f3259\\images/8e8a7a14749d0b2e48de3d10e2e80063f17b165ad921c8afc0623f0850
0f3259.png'}
Traceback (most recent call last):
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1702, in data_generator
    load_image_gt(dataset, config, image_id, augment=augment,
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\model.py", line 1287, in load_image_gt
    mask = utils.minimize_mask(bbox, mask, config.MINI_MASK_SHAPE)
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\mrcnn\utils.py", line 541, in minimize_mask
    m = skimage.transform.resize(m, mini_shape, order=1, mode="constant")
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\transform\_warps.py", line 160, in resize
  File "C:\Users\Hariket Sheth\AppData\Local\Programs\Python\Python310\lib\s
ite-packages\skimage\_shared\utils.py", line 724, in _validate_interpolation
_order
ValueError: Input image dtype is bool. Interpolation is not defined with boo
l data type. Please set order to 0 or explicitely cast input image to anothe
r data type.

---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Input In [150], in <cell line: 2>()
      1 # Get Next Image
```

```
      2 if random_rois:
      3     [normalized_images, image_meta, rpn_match, rpn_bbox, gt_class_
ids, gt_boxes, gt_masks, rpn_rois, rois], \
----> 4     [mrcnn_class_ids, mrcnn_bbox, mrcnn_mask] = next(g)
      6     log("rois", rois)
      7     log("mrcnn_class_ids", mrcnn_class_ids)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\mrcnn\mod
el.py:1702, in data_generator(dataset, config, shuffle, augment, augmentat
ion, random_rois, batch_size, detection_targets)
   1699 # Get GT bounding boxes and masks for image.
   1700 image_id = image_ids[image_index]
   1701 image, image_meta, gt_class_ids, gt_boxes, gt_masks = \
-> 1702     load_image_gt(dataset, config, image_id, augment=augment,
   1703                   augmentation=augmentation,
   1704                   use_mini_mask=config.USE_MINI_MASK)
   1706 # Skip images that have no instances. This can happen in cases
   1707 # where we train on a subset of classes and the image doesn't
   1708 # have any of the classes we care about.
   1709 if not np.any(gt_class_ids > 0):

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\mrcnn\mod
el.py:1287, in load_image_gt(dataset, config, image_id, augment, augmentat
ion, use_mini_mask)
   1285 # Resize masks to smaller size to reduce memory usage
   1286 if use_mini_mask:
-> 1287     mask = utils.minimize_mask(bbox, mask, config.MINI_MASK_SHAPE)
   1289 # Image meta data
   1290 image_meta = compose_image_meta(image_id, original_shape, image.sh
ape,
   1291                                  window, scale, active_class_ids)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\mrcnn\uti
ls.py:541, in minimize_mask(bbox, mask, mini_shape)
    539         raise Exception("Invalid bounding box with area of zero")
    540     # Resize with bilinear interpolation
--> 541     m = skimage.transform.resize(m, mini_shape, order=1, mode="con
stant")
    542     mini_mask[:, :, i] = np.around(m).astype(np.bool)
    543 return mini_mask

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\skimage\t
ransform\_warps.py:160, in resize(image, output_shape, order, mode, cval,
 clip, preserve_range, anti_aliasing, anti_aliasing_sigma)
    157     raise ValueError("anti_aliasing must be False for boolean imag
es")
    159 factors = np.divide(input_shape, output_shape)
--> 160 order = _validate_interpolation_order(input_type, order)
    161 if order > 0:
    162     image = convert_to_float(image, preserve_range)

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\skimage\_
shared\utils.py:724, in _validate_interpolation_order(image_dtype, order)
    720     raise ValueError("Spline interpolation order has to be in the
"
    721                      "range 0-5.")
    723 if image_dtype == bool and order != 0:
--> 724     raise ValueError(
    725         "Input image dtype is bool. Interpolation is not defined "
    726         "with bool data type. Please set order to 0 or explicitel
y "
```

```
    727         "cast input image to another data type.")
    729 return order

ValueError: Input image dtype is bool. Interpolation is not defined with b
ool data type. Please set order to 0 or explicitely cast input image to an
other data type.
```

In [151]:

```
b = 0

# Restore original image (reverse normalization)
sample_image = modellib.unmold_image(normalized_images[b], config)

# Compute anchor shifts.
indices = np.where(rpn_match[b] == 1)[0]
refined_anchors = utils.apply_box_deltas(anchors[indices], rpn_bbox[b, :len(indices)] * con
log("anchors", anchors)
log("refined_anchors", refined_anchors)

# Get list of positive anchors
positive_anchor_ids = np.where(rpn_match[b] == 1)[0]
print("Positive anchors: {}".format(len(positive_anchor_ids)))
negative_anchor_ids = np.where(rpn_match[b] == -1)[0]
print("Negative anchors: {}".format(len(negative_anchor_ids)))
neutral_anchor_ids = np.where(rpn_match[b] == 0)[0]
print("Neutral anchors: {}".format(len(neutral_anchor_ids)))

# ROI breakdown by class
for c, n in zip(dataset.class_names, np.bincount(mrcnn_class_ids[b].flatten())):
    if n:
        print("{:23}: {}".format(c[:20], n))

# Show positive anchors
fig, ax = plt.subplots(1, figsize=(16, 16))
visualize.draw_boxes(sample_image, boxes=anchors[positive_anchor_ids],
                     refined_boxes=refined_anchors, ax=ax)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [151], in <cell line: 4>()
      1 b = 0
      3 # Restore original image (reverse normalization)
----> 4 sample_image = modellib.unmold_image(normalized_images[b], config)
      6 # Compute anchor shifts.
      7 indices = np.where(rpn_match[b] == 1)[0]

NameError: name 'normalized_images' is not defined
```

In [ ]:

```
# Show negative anchors
visualize.draw_boxes(sample_image, boxes=anchors[negative_anchor_ids])
```

In [ ]:

```
# Show neutral anchors. They don't contribute to training.
visualize.draw_boxes(sample_image, boxes=anchors[np.random.choice(neutral_anchor_ids, 100)]
```

ROIs

Typically, the RPN network generates region proposals (a.k.a. Regions of Interest, or ROIs). The data generator has the ability to generate proposals as well for illustration and testing purposes. These are controlled by the `random_rois` parameter.

In [ ]:

```
if random_rois:
    # Class aware bboxes
    bbox_specific = mrcnn_bbox[b, np.arange(mrcnn_bbox.shape[1]), mrcnn_class_ids[b], :]

    # Refined ROIs
    refined_rois = utils.apply_box_deltas(rois[b].astype(np.float32), bbox_specific[:,:4] *

    # Class aware masks
    mask_specific = mrcnn_mask[b, np.arange(mrcnn_mask.shape[1]), :, :, mrcnn_class_ids[b]]

    visualize.draw_rois(sample_image, rois[b], refined_rois, mask_specific, mrcnn_class_ids

    # Any repeated ROIs?
    rows = np.ascontiguousarray(rois[b]).view(np.dtype((np.void, rois.dtype.itemsize * rois
    _, idx = np.unique(rows, return_index=True)
    print("Unique ROIs: {} out of {}".format(len(idx), rois.shape[1]))
```

In [ ]:

```
if random_rois:
    # Dispalay ROIs and corresponding masks and bounding boxes
    ids = random.sample(range(rois.shape[1]), 8)

    images = []
    titles = []
    for i in ids:
        image = visualize.draw_box(sample_image.copy(), rois[b,i,:4].astype(np.int32), [255
        image = visualize.draw_box(image, refined_rois[i].astype(np.int64), [0, 255, 0])
        images.append(image)
        titles.append("ROI {}".format(i))
        images.append(mask_specific[i] * 255)
        titles.append(dataset.class_names[mrcnn_class_ids[b,i]][:20])

    display_images(images, titles, cols=4, cmap="Blues", interpolation="none")
```

In [ ]:

```
# Check ratio of positive ROIs in a set of images.
if random_rois:
    limit = 10
    temp_g = modellib.data_generator(
        dataset, crop_config, shuffle=True, random_rois=10000,
        batch_size=1, detection_targets=True)
    total = 0
    for i in range(limit):
        _, [ids, _, _] = next(temp_g)
        positive_rois = np.sum(ids[0] > 0)
        total += positive_rois
        print("{:5} {:5.2f}".format(positive_rois, positive_rois/ids.shape[1]))
    print("Average percent: {:.2f}".format(total/(limit*ids.shape[1])))
```

In [ ]:

```
```