DAFTAR PUSTAKA

Anaconda Inc. (2021) *Anaconda | The World's Most Popular Data Science Platform*, *Anaconda*. Available at: https://www.anaconda.com/ (Accessed: 1 July 2023).

Bhatti, H. M. A. *et al.* (2020) 'Multi-detection and Segmentation of Breast Lesions Based on Mask RCNN-FPN', *Proceedings - 2020 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2020*, pp. 2698–2704. doi: 10.1109/BIBM49941.2020.9313170.

C. Arteta, V. Lempitsky, J. A. Noble, and A. Zisserman, "Learning to detect partially overlapping instances," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 3230–3237, 2013, doi: 10.1109/CVPR.2013.415.

C. C. Tran, Di. T. Nguyen, H. D. Le, Q. B. Truong, and Q. Di. Truong, "Automatic dragon fruit counting using adaptive thresholds for image segmentation and shape analysis," *2017 4th NAFOSTED Conf. Inf. Comput. Sci. NICS 2017 - Proc.*, vol. 2017-Janua, pp. 132–137, 2017, doi: 10.1109/NAFOSTED.2017.8108052.

Elgendy, M, (2020) Deep Learning for Vision Systems, Manning Publications Co, Grandini, M,, Bagli, E,, Visani, G,, 2020, Metrics for Multi-Class Classification: an Overview

Fajri, L.R.H.A. (2022, Januari 3). Artificial Neural Network. Universitas STEKOM (Sains &Teknologi Komputer). http://sistem-informasis1.stekom.ac.id/informasi/baca/Artificial-NeuralNetwork/b1c26e9347ef547ff06845ca38cc443aedc4fa86

He, K. *et al.* (2017) 'Mask R-CNN', *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob, pp. 2980–2988. doi: 10.1109/ICCV.2017.322.

J. Ni, Z. Khan, S. Wang, K. Wang, and S. K. Haider, "Automatic detection and counting of circular shaped overlapped objects using circular hough transform and contour detection," *Proc. World Congr. Intell. Control Autom.*, vol. 2016-Septe, no. Kylx15 0496, pp. 2902–2906, 2016, doi: 10.1109/WCICA.2016.7578268.

K. Abhinav, J. S. Chauhan, and D. Sarkar, "Image segmentation of multi-shaped overlapping objects," *VISIGRAPP 2018 - Proc. 13th Int. Jt. Conf. Comput. Vision, Comput. Graph. Theory Appl.*, vol. 4, no. Visigrapp, pp. 410–418, 2018, 220/0006628404100418.

., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553),

Mahmood, A. *et al.* (2020) 'Automatic hierarchical classification of kelps using deep residual features', *Sensors (Switzerland)*, 20(2), pp. 1–20. doi: 10.3390/s20020447.

M. Swaraj Raman and M. Sukanya, "A novel labelling algorithm for object counting," *2012 3rd Int. Conf. Comput. Commun. Netw. Technol. ICCCNT 2012*, no. July, 2012, doi: 10.1109/ICCCNT.2012.6395904.

N. Unde, "Implementation of Smart Shopping Cart Using RFID," no. 6, 2015.

P. J. Ramos, F. A. Prieto, E. C. Montoya, and C. E. Oliveros, "Automatic fruit count on coffee branches using computer vision," *Comput. Electron. Agric.*, vol. 137, pp. 9–22, 2017, doi: 10.1016/j.compag.2017.03.010.

Priyanto Hidayatullah (2021) Buku Sakti Deep Learning, Stunning Vision Al Academy

Randles, B, M, et al, (2017) 'Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study', Proceedings of the ACM/IEEE Joint Conference on Digital Libraries, doi: 10,1109/JCDL,2017,7991618,

R. Ardianto, T. Rivanie, Y. Alkhalifi, F. S. Nugraha, and W. Gata, "Jurnal Ilmu Komputer dan Informasi (Journal of Computer Science and Information)," *J. Comput. Sci. Inf.*, vol. 2, no. sentiment analysis, p. 13, 2020.

R. Hussin, M. R. Juhari, N. W. Kang, R. C. Ismail, and A. Kamarudin, "Digital image processing techniques for object detection from complex background image," *Procedia Eng.*, vol. 41, no. Iris, pp. 340–344, 2012, doi: 10.1016/j.proeng.2012.07.182.

R. M. Akbar and N. Sunarmi, "Pengenalan Barang Pada Kereta Belanja Menggunakan Metode Scale Invariant Feature Transform (SIFT)," *J. Teknol. Inf. dan Ilmu Komput.*, vol. 5, no. 6, p. 667, 2018, doi: 10.25126/jtiik.2018561046.

Sanjaya, J,, Ayub, M,, 2020, Augmentasi Data Pengenalan Citra Mobil Menggunakan Pendekatan Random Crop, Rotate, dan Mixup, JuTISI 6, https://doi,org/10,28932/jutisi,v6i2,2688

Seema Singh (2018) Cousins of Artificial Intelligence | by Seema Singh | Towards Data Science, Available at: https://towardsdatascience,com/cousins-of-artificial-ce-dda4edc27b55 (Accessed: 18 June 2023)

er, E., Long, J. and Darrell, T. (2017) 'Fully Convolutional Networks for Segmentation', *IEEE Transactions on Pattern Analysis and Machine*

Intelligence, 39(4), pp. 640–651. doi: 10.1109/TPAMI.2016.2572683.

LAMPIRAN

1.1. Lampiran *Source Code*

A. Source Code Training

```
"""
Mask R-CNN
Train on the toy Balloon dataset and implement color splash
effect.

Copyright (c) 2018 Matterport, Inc.
Licensed under the MIT License (see LICENSE for details)
Written by Waleed Abdulla

------------------------------------------------------------

Usage: import the module (see Jupyter notebooks for examples),
or run from
 the command line as such:

 # Train a new model starting from pre-trained COCO weights
 python3 balloon.py train --dataset=/path/to/balloon/dataset --
weights=coco

 # Resume training a model that you had trained earlier
 python3 balloon.py train --dataset=/path/to/balloon/dataset --
weights=last

 # Train a new model starting from ImageNet weights
 python3 balloon.py train --dataset=/path/to/balloon/dataset --
weights=imagenet

 # Apply color splash to an image
 python3 balloon.py splash --weights=/path/to/weights/file.h5 -
-image=<URL or path to file>

 # Apply color splash to video using the last weights you
trained
 python3 balloon.py splash --weights=last --video=<URL or path
to file>
"""

import os
import sys
import json
import datetime
import numpy as np
import skimage.draw
import wandb
    mport matplotlib.pyplot as plt
    mport keras

    Root directory of the project
)OT_DIR = "C:/Users/User/Armadi/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 model as modellib, utils

# Path to trained weights file
COCO_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")

# Directory to save logs and model checkpoints, if not provided
# through the command line argument --logs
DEFAULT_LOGS_DIR = os.path.join(ROOT_DIR, "logs")

############################################################
# Configurations
############################################################


class CustomConfig(Config):
 """Configuration for training on the custom dataset.
 Derives from the base Config class and overrides some values.
 """
 # Give the configuration a recognizable name
 NAME = "names"

 # We use a GPU with 12GB memory, which can fit two images.
 # Adjust down if you use a smaller GPU.
 IMAGES_PER_GPU = 1

 # Number of classes (including background)
 NUM_CLASSES = 1 + 6 # Background + Object

 # Number of training steps per epoch
 STEPS_PER_EPOCH = 200

 # Skip detections with < 90% confidence
 DETECTION_MIN_CONFIDENCE = 0.9


 def get_config_dict(self):
 """Return Configuration values as a dictionary for the sake of
 syncing with wandb"""
 d = {}
 for a in dir(self):
 if not a.startswith("__") and not callable(getattr(self, a)):
 d[a] = getattr(self, a)
 return d

############################################################
# WANDB
############################################################

run = wandb.init(project="overlapp")
 config = CustomConfig()

 onfig_dict = _config.get_config_dict()
 onfigs_of_interest = ['BACKBONE', 'GRADIENT_CLIP_NORM',
 EARNING_MOMENTUM', 'LEARNING_RATE',
 'WEIGHT_DECAY', 'STEPS_PER_EPOCH']
```

```
99.
100.      wandb.log({k: config_dict[k] for k in
   configs_of_interest})
101.
102.
103.      def fig_to_array(fig):
104.       fig.canvas.draw()
105.       w, h = fig.canvas.get_width_height()
106.       buf = np.fromstring(fig.canvas.tostring_argb(),
   dtype=np.uint8)
107.       buf.shape = (w, h, 4)
108.       buf = np.roll(buf, 3, axis=2)
109.       return buf
110.
111.      class ImageCallback(keras.callbacks.Callback):
112.       def __init__(self, run, dataset_val,
   dataset_train,infer_config,log_dir):
113.       super(ImageCallback, self).__init__()
114.       self.run = run
115.       self.dataset_val = dataset_val
116.       self.dataset_train = dataset_train
117.       self.image_ids = dataset_val.image_ids[:3]
118.       self.infer_config = infer_config
119.       self.log_dir=log_dir
120.       self.inf_model = modellib.MaskRCNN(mode="inference",
   config=self.infer_config ,
121.       model_dir=log_dir,callbacks=[])
122.
123.       def load_curr_model(self):
124.       model_path = self.inf_model.find_last()[1]
125.       self.inf_model.load_weights(model_path, by_name=True)
126.
127.       def predict_image(self, image_id):
128.       original_image, image_meta, gt_class_id, gt_bbox,
   gt_mask = load_image_gt(
129.       self.dataset_val, _config, image_id,
   use_mini_mask=False)
130.       _, ax = plt.subplots(figsize=(16, 16))
131.
132.       # Run detection
133.       results = self.inf_model.detect([original_image])
134.
135.       # Visualize results
136.       r = results[0]
137.       visualize.display_instances(original_image, r['rois'],
   r['masks'], r['class_ids'],
138.       self.dataset_val.class_names,
   r['scores'],figsize=(16,16),
139.       ax=ax)
140.       return fig_to_array(ax.figure)
141.
142.       def on_epoch_end(self, epoch, logs):
143.       print("Uploading images to wandb...")
          self.load_curr_model();
          predicted_images = [self.predict_image(i) for i in
    elf.image_ids]
          wandb.log({"img_segmentations":[
          wandb.Image(
          scipy.misc.imresize(img, 50),
```

```
149.        caption="SampleImage",
150.        mode='RGBA') for img in predicted_images]})
151.
152.      class PerformanceCallback(keras.callbacks.Callback):
153.        def __init__(self, run):
154.        self.run = run
155.        def on_epoch_end(self, epoch, logs):
156.        print("Uploading metrics to wandb...")
157.        wandb.log(logs)
158.
159.      ############################################################
    ####
160.      # Dataset
161.      ############################################################
    ####
162.
163.      class CustomDataset(utils.Dataset):
164.
165.        def load_custom(self, dataset_dir, subset):
166.        """Load a subset of the Dog-Cat dataset.
167.        dataset_dir: Root directory of the dataset.
168.        subset: Subset to load: train or val
169.        """
170.        # Add classes. We have only one class to add.
171.
172.        self.add_class("names", 1, "Oreo Mini")
173.        self.add_class("names", 2, "Pringles")
174.        self.add_class("names", 3, "SilverQueen")
175.        self.add_class("names", 4, "Milo Sereal" )
176.        self.add_class("names", 5, "Ultra Milk")
177.        self.add_class("names", 6, "Oreo")
178.
179.
180.        # Train or validation dataset?
181.        assert subset in ["train", "val"]
182.        dataset_dir = os.path.join(dataset_dir, subset)
183.
184.        # Load annotations
185.        # VGG Image Annotator (up to version 1.6) saves each
    image in the form:
186.        # { 'filename': '28503151_5b5b7ec140_b.jpg',
187.        # 'regions': {
188.        # '0': {
189.        # 'region_attributes': {},
190.        # 'shape_attributes': {
191.        # 'all_points_x': [...],
192.        # 'all_points_y': [...],
193.        # 'name': 'polygon'}},
194.        # ... more regions ...
195.        # },
196.        # 'size': 100202
197.        # }
198.        # We mostly care about the x and y coordinates of each
    egion
            # Note: In VIA 2.0, regions was changed from a dict to
     list.
            annotations = json.load(open(os.path.join(dataset_dir,
    ria_region_data.json")))
```

```
        annotations = list(annotations.values()) # don't need
    the dict keys

        # The VIA tool saves images in the JSON even if they
    don't have any
        # annotations. Skip unannotated images.
        annotations = [a for a in annotations if a['regions']]

        # Add images
        for a in annotations:
        # Get the x, y coordinaets of points of the polygons
    that make up
        # the outline of each object instance. These are stores
    in the
        # shape_attributes (see json format above)
        # The if condition is needed to support VIA versions
    1.x and 2.x.
        polygons=[]
        objects=[]
        for r in a['regions'].values():
        polygons.append(r['shape_attributes'])
        # print("polygons=", polygons)
        objects.append(r['region_attributes'])
        # print("multi_numbers=", multi_numbers)
        #polygons = [r['shape_attributes'] for r in
    a['regions'].values()]
        #objects = [s['region_attributes'] for s in
    a['regions'].values()]
        class_ids = [int(n['names']) for n in objects]
        # load_mask() needs the image size to convert polygons
    to masks.
        # Unfortunately, VIA doesn't include it in JSON, so we
    must read
        # the image. This is only managable since the dataset
    is tiny.
        # print("multi_numbers=", multi_numbers)
        # num_ids = [n for n in
    multi_numbers['number'].values()]
        # for n in multi_numbers:

        image_path = os.path.join(dataset_dir, a['filename'])
        image = skimage.io.imread(image_path)
        height, width = image.shape[:2]

        self.add_image(
        "names",
        image_id=a['filename'], # use file name as a unique
    image id
        path=image_path,
        width=width, height=height,
        polygons=polygons,
        class_ids=class_ids)

        def load_mask(self, image_id):
        """Generate instance masks for an image.
        Returns:
        masks: A bool array of shape [height, width, instance
    ount] with
        one mask per instance.
```

```
        class_ids: a 1D array of class IDs of the instance masks.
        """
        # If not a balloon dataset image, delegate to parent class.
        image_info = self.image_info[image_id]
        if image_info["source"] != "names":
        return super(self.__class__, self).load_mask(image_id)
        class_ids = image_info['class_ids']
        # Convert polygons to a bitmap mask of shape
        # [height, width, instance_count]
        info = self.image_info[image_id]
        mask = np.zeros([info["height"], info["width"], len(info["polygons"])],
        dtype=np.uint8)
        for i, p in enumerate(info["polygons"]):
        # Get indexes of pixels inside the polygon and set them to 1
        rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x'])
        mask[rr, cc, i] = 1

        # Return mask, and array of class IDs of each instance. Since we have
        # one class ID only, we return an array of 1s
        #class_ids=np.array([self.class_names.index(shapes[0])])
        #print("info['class_ids']=", info['class_ids'])
        class_ids = np.array(class_ids, dtype=np.int32)
        return mask, class_ids#[mask.shape[-1]] #np.ones([mask.shape[-1]], dtype=np.int32)#class_ids.astype(np.int32)

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


      def train(model):
        """Train the model."""
        # Training dataset.
        dataset_train = CustomDataset()
        dataset_train.load_custom(args.dataset, "train")
        dataset_train.prepare()

        # Validation dataset
        dataset_val = CustomDataset()
        dataset_val.load_custom(args.dataset, "val")
        dataset_val.prepare()

        # *** This training schedule is an example. Update to
 our needs ***
        # Since we're using a very small dataset, and starting
 rom
```

```
294.        # COCO trained weights, we don't need to train too
    long. Also,
295.        # no need to train all layers, just the heads should do
    it.
296.        print("Training network heads")
297.        model.train(dataset_train, dataset_val,
298.        learning_rate=config.LEARNING_RATE,
299.        epochs=250,
300.        layers='heads')
301.      def color_splash(image, mask):
302.        """Apply color splash effect.
303.        image: RGB image [height, width, 3]
304.        mask: instance segmentation mask [height, width,
    instance count]
305.        Returns result image.
306.        """
307.        # Make a grayscale copy of the image. The grayscale
    copy still
308.        # has 3 RGB channels, though.
309.        gray =
    skimage.color.gray2rgb(skimage.color.rgb2gray(image)) * 255
310.        # Copy color pixels from the original color image where
    mask is set
311.        if mask.shape[-1] > 0:
312.        # We're treating all instances as one, so collapse the
    mask into one layer
313.        mask = (np.sum(mask, -1, keepdims=True) >= 1)
314.        splash = np.where(mask, image, gray).astype(np.uint8)
315.        else:
316.        splash = gray.astype(np.uint8)
317.        return splash
318.      def detect_and_color_splash(model, image_path=None,
    video_path=None):
319.        assert image_path or video_path
320.        # Image or video?
321.        if image_path:
322.        # Run model detection and generate the color splash
    effect
323.        print("Running on {}".format(args.image))
324.        # Read image
325.        image = skimage.io.imread(args.image)
326.        # Detect objects
327.        r = model.detect([image], verbose=1)[0]
328.        # Color splash
329.        splash = color_splash(image, r['masks'])
330.        # Save output
331.        file_name =
    "splash_{:%Y%m%dT%H%M%S}.png".format(datetime.datetime.now())
332.        skimage.io.imsave(file_name, splash)
333.        elif video_path:
334.        import cv2
335.        # Video capture
336.        vcapture = cv2.VideoCapture(video_path)
            width = int(vcapture.get(cv2.CAP_PROP_FRAME_WIDTH))
            height = int(vcapture.get(cv2.CAP_PROP_FRAME_HEIGHT))
            fps = vcapture.get(cv2.CAP_PROP_FPS)

            # Define codec and create video writer
```

```
342.        file_name =
    "splash_{:%Y%m%dT%H%M%S}.avi".format(datetime.datetime.now())
343.        vwriter = cv2.VideoWriter(file_name,
344.        cv2.VideoWriter_fourcc(*'MJPG'),
345.        fps, (width, height))
346.
347.        count = 0
348.        success = True
349.        while success:
350.        print("frame: ", count)
351.        # Read next image
352.        success, image = vcapture.read()
353.        if success:
354.        # OpenCV returns images as BGR, convert to RGB
355.        image = image[..., ::-1]
356.        # Detect objects
357.        r = model.detect([image], verbose=0)[0]
358.        # Color splash
359.        splash = color_splash(image, r['masks'])
360.        # RGB -> BGR to save image to video
361.        splash = splash[..., ::-1]
362.        # Add image to video writer
363.        vwriter.write(splash)
364.        count += 1
365.        vwriter.release()
366.        print("Saved to ", file_name)
367.
368.
369.      ############################################################
    ####
370.      # Training
371.      ############################################################
    ####
372.
373.      if __name__ == '__main__':
374.        import argparse
375.
376.        # Parse command line arguments
377.        parser = argparse.ArgumentParser(
378.        description='Train Mask R-CNN to detect balloons.')
379.        parser.add_argument("command",
380.        metavar="<command>",
381.        help="'train' or 'splash'")
382.        parser.add_argument('--dataset', required=False,
383.        metavar="/path/to/balloon/dataset/",
384.        help='Directory of the Balloon dataset')
385.        parser.add_argument('--weights', required=True,
386.        metavar="/path/to/weights.h5",
387.        help="Path to weights .h5 file or 'coco'")
388.        parser.add_argument('--logs', required=False,
389.        default=DEFAULT_LOGS_DIR,
390.        metavar="/path/to/logs/",
391.        help='Logs and checkpoints directory (default=logs/)')
            parser.add_argument('--image', required=False,
            metavar="path or URL to image",
            help='Image to apply the color splash effect on')
            parser.add_argument('--video', required=False,
            metavar="path or URL to video",
            help='Video to apply the color splash effect on')
```

```
398.        args = parser.parse_args()
399.
400.        # Validate arguments
401.        if args.command == "train":
402.        assert args.dataset, "Argument --dataset is required
    for training"
403.        elif args.command == "splash":
404.        assert args.image or args.video,\
405.        "Provide --image or --video to apply color splash"
406.
407.        print("Weights: ", args.weights)
408.        print("Dataset: ", args.dataset)
409.        print("Logs: ", args.logs)
410.
411.        # Configurations
412.        if args.command == "train":
413.        config = CustomConfig()
414.        else:
415.        class InferenceConfig(CustomConfig):
416.        # Set batch size to 1 since we'll be running inference
    on
417.        # one image at a time. Batch size = GPU_COUNT *
    IMAGES_PER_GPU
418.        GPU_COUNT = 1
419.        IMAGES_PER_GPU = 1
420.        config = InferenceConfig()
421.        config.display()
422.
423.        # Create model
424.        if args.command == "train":
425.        model = modellib.MaskRCNN(mode="training",
    config=config,
426.        model_dir=args.logs)
427.        else:
428.        model = modellib.MaskRCNN(mode="inference",
    config=config,
429.        model_dir=args.logs)
430.
431.        # Select weights file to load
432.        if args.weights.lower() == "coco":
433.        weights_path = COCO_WEIGHTS_PATH
434.        # Download weights file
435.        if not os.path.exists(weights_path):
436.        utils.download_trained_weights(weights_path)
437.        elif args.weights.lower() == "last":
438.        # Find last trained weights
439.        weights_path = model.find_last()
440.        elif args.weights.lower() == "imagenet":
441.        # Start from ImageNet trained weights
442.        weights_path = model.get_imagenet_weights()
443.        else:
444.        weights_path = args.weights
445.
            # Load weights
            print("Loading weights ", weights_path)
            if args.weights.lower() == "coco":
            # Exclude the last layers because they require a
        atching
            # number of classes
```

```
451.        model.load_weights(weights_path, by_name=True,
    exclude=[
452.        "mrcnn_class_logits", "mrcnn_bbox_fc",
453.        "mrcnn_bbox", "mrcnn_mask"])
454.        else:
455.        model.load_weights(weights_path, by_name=True)
456.
457.        # Train or evaluate
458.        if args.command == "train":
459.        train(model)
460.        elif args.command == "splash":
461.        detect_and_color_splash(model, image_path=args.image,
462.        video_path=args.video)
463.        else:
464.        print("'{}' is not recognized. "
465.        "Use 'train' or 'splash'".format(args.command))
```

A. *Source Code* Testing

```
1. import warnings
2. warnings.filterwarnings('ignore')
3. import os
4. import sys
5.
6. # Root directory of the project
7. ROOT_DIR = "C:/Users/User/Armadi/Mask_RCNN/"
8.
9. # Import Mask RCNN
10.sys.path.append(ROOT_DIR)  # To find local version of the
    library
11.
12.import json
13.import datetime
14.import numpy as np
15.import skimage.draw
16.import cv2
17.import random
18.import math
19.import re
20.import time
21.import tensorflow as tf
22.import matplotlib.pyplot as plt
23.import matplotlib.patches as patches
24.import matplotlib.image as mpimg
25.from mrcnn import utils
26.from mrcnn import visualize
27.from mrcnn.visualize import display_images
28.from mrcnn.visualize import display_instances
29.#from mrcnn.visualize import draw_text
30.import mrcnn.model as modellib
31.from mrcnn.model import log
32.from mrcnn.config import Config
    :om mrcnn import model as modellib, utils

    import custom
    EFAULT_LOGS_DIR = os.path.join(ROOT_DIR, "logs")

    )DEL_DIR = os.path.join(ROOT_DIR, "logs")
```

```
39.
40.WEIGHTS_PATH =
   "C:/Users/User/Armadi/Mask_RCNN/logs/names20230315T0009/mask_rc
   nn_names_0250.h5"    # change it
41.class CustomConfig(Config):
42.    """Configuration for training on the custom  dataset.
43.    Derives from the base Config class and overrides some
   values.
44.    """
45.    # Give the configuration a recognizable name
46.    NAME = "names"
47.
48.    IMAGES_PER_GPU = 1
49.
50.    NUM_CLASSES = 1 + 6  # Background + Car and truck
51.
52.    # Number of training steps per epoch
53.    STEPS_PER_EPOCH = 200
54.
55.    # Skip detections with < 90% confidence
56.    DETECTION_MIN_CONFIDENCE = 0.7
57.
58.    USE_MINI_MASK = False
59.    MINI_MASK_SHAPE = (56, 56)  # (height, width) of the mini-
   mask
60.# Code for Customdataset class. Same code is present in
   custom.py file also
61.class CustomDataset(utils.Dataset):
62.
63.    def load_custom(self, dataset_dir, subset):
64.        """Load a subset of the Dog-Cat dataset.
65.        dataset_dir: Root directory of the dataset.
66.        subset: Subset to load: train or val
67.        """
68.        # Add classes. We have only one class to add.
69.
70.        self.add_class("names", 1, "Oreo Mini")
71.        self.add_class("names", 2, "Pringles")
72.        self.add_class("names", 3, "SilverQueen")
73.        self.add_class("names", 4, "Milo Sereal" )
74.        self.add_class("names", 5, "Ultra Milk")
75.        self.add_class("names", 6, "Oreo")
76.
77.        # Train or validation dataset?
78.        assert subset in ["train", "val"]
79.        dataset_dir = os.path.join(dataset_dir, subset)
80.
81.        annotations = json.load(open(os.path.join(dataset_dir,
   "via_region_data.json")))
82.        annotations = list(annotations.values())  # don't need
   the dict keys
83.
84.        annotations = [a for a in annotations if a['regions']]

           # Add images
           for a in annotations:
               polygons=[]
               objects=[]
               for r in a['regions'].values():
```

```
                polygons.append(r['shape_attributes'])
            # print("polygons=", polygons)
                objects.append(r['region_attributes'])
            class_ids = [int(n['names']) for n in objects]

            image_path = os.path.join(dataset_dir, a['filename'])
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]

                  self.add_image(
                      "names",
                      image_id=a['filename'],  # use file name as a unique image id
                      path=image_path,
                      width=width, height=height,
                      polygons=polygons,
                           class_ids=class_ids)

        def load_mask(self, image_id):
            """Generate instance masks for an image.
           Returns:
            masks: A bool array of shape [height, width, instance count] with
                one mask per instance.
            class_ids: a 1D array of class IDs of the instance masks.
            """
            # If not a balloon dataset image, delegate to parent class.
            image_info = self.image_info[image_id]
            if image_info["source"] != "names":
                return super(self.__class__, self).load_mask(image_id)
            class_ids = image_info['class_ids']
            # Convert polygons to a bitmap mask of shape
            # [height, width, instance_count]
            info = self.image_info[image_id]
            mask = np.zeros([info["height"], info["width"], len(info["polygons"])],
                            dtype=np.uint8)
            for i, p in enumerate(info["polygons"]):
                # Get indexes of pixels inside the polygon and set them to 1
                rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x'])
                mask[rr, cc, i] = 1

            # Return mask, and array of class IDs of each instance. Since we have
            # one class ID only, we return an array of 1s
    #class_ids=np.array([self.class_names.index(shapes[0])])
            #print("info['class_ids']=", info['class_ids'])
            class_ids = np.array(class_ids, dtype=np.int32)
            return mask, class_ids#[mask.shape[-1]]
    np.ones([mask.shape[-1]],
    dtype=np.int32)#class_ids.astype(np.int32)
        def image_reference(self, image_id):
```

```
            """Return the path of the image."""
            info = self.image_info[image_id]
            if info["source"] == "names":
                return info["path"]
            else:
                super(self.__class__, self).image_reference(image_id)
    TEST_MODE = "inference"
    ROOT_DIR = "C:\\Users\\User\\Armadi\\Mask_RCNN\\samples\\product\\dataset"

    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
    # Load validation dataset
    # Must call before using the dataset
    CUSTOM_DIR = "C:\\Users\\User\\Armadi\\Mask_RCNN\\samples\\product\\dataset"
    dataset = CustomDataset()
    dataset.load_custom(CUSTOM_DIR, "val")
    dataset.prepare()
    print("Images: {}\nClasses: {}".format(len(dataset.image_ids), dataset.class_names))
    config = CustomConfig()
    #LOAD MODEL. Create model in inference mode
    model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR, config=config)
    # Load COCO weights Or, load the last model you trained
    weights_path = WEIGHTS_PATH
    # Load weights
    print("Loading weights ", weights_path)
    model.load_weights(weights_path, by_name=True)
    model
    from collections import defaultdict
    # Path to the new image
    image_path = 'C:\\Users\\User\\Armadi\\Mask_RCNN\\samples\\product\\testBaru\\880cm.jpg'

    # Load the imag
    image = skimage.io.imread(image_path)

    # Run object detection
    results = model.detect([image], verbose=0)
    r = results[0]

    # Count the number of objects detected
    num_objects = r['rois'].shape[0]

    print("Number of objects detected: ", num_objects)

    class_names = dataset.class_names

    # Display the class names of the detected objects
```

```
185.      for i in range(num_objects):
186.          class_name = class_names[r['class_ids'][i]]
187.          print("Detected object #{}: {}".format(i+1,
   class_name))
188.
189.      # Visualize the detected objects on the imageS
190.      visualize.display_instances(image, r['rois'],
   r['masks'], r['class_ids'],
191.                                  class_names, r['scores'])
```