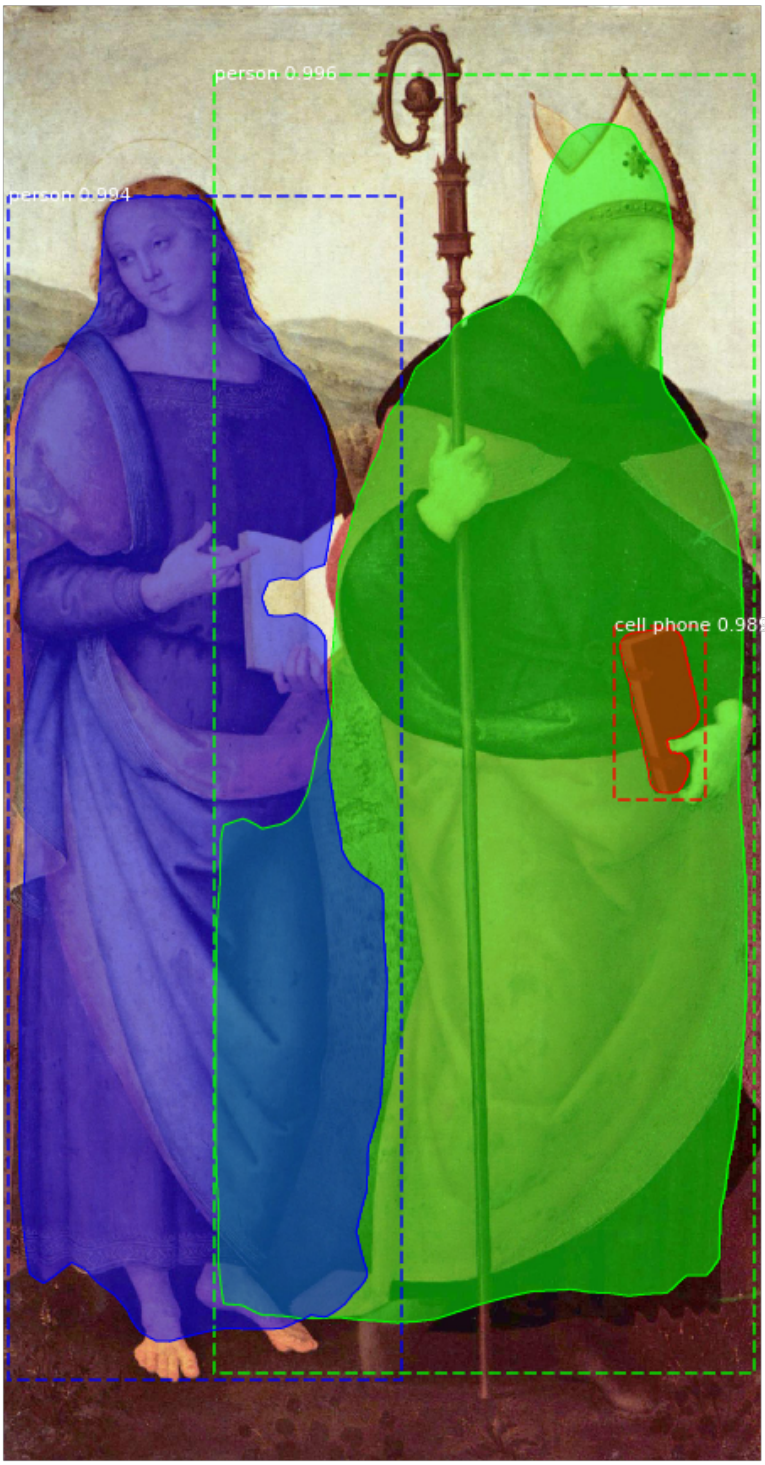
In [ ]:
```
import os
import sys
import random
import math
import numpy as np
import skimage.io
import matplotlib
import matplotlib.pyplot as plt
import mrcnn.model as modellib
sys.path.append(os.path.join("samples/coco/"))
import coco
from mrcnn import utils
from mrcnn import visualize

%matplotlib inline

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

# Local path to trained weights file
COCO_MODEL_PATH = os.path.join("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)

# Directory of images to run detection on
IMAGE_DIR = os.path.join("images_examp")
```

In [2]:
```
class InferenceConfig(coco.CocoConfig):
    # Set batch size to 1 since we'll be running inference on
    # one image at a time. Batch size = GPU_COUNT * IMAGES_PER_GPU
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1

config = InferenceConfig()
```

In [ ]:
```
# Create model object in inference mode.
model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR, config=confi
g)

# Load weights trained on MS-COCO
model.load_weights(COCO_MODEL_PATH, by_name=True)
```

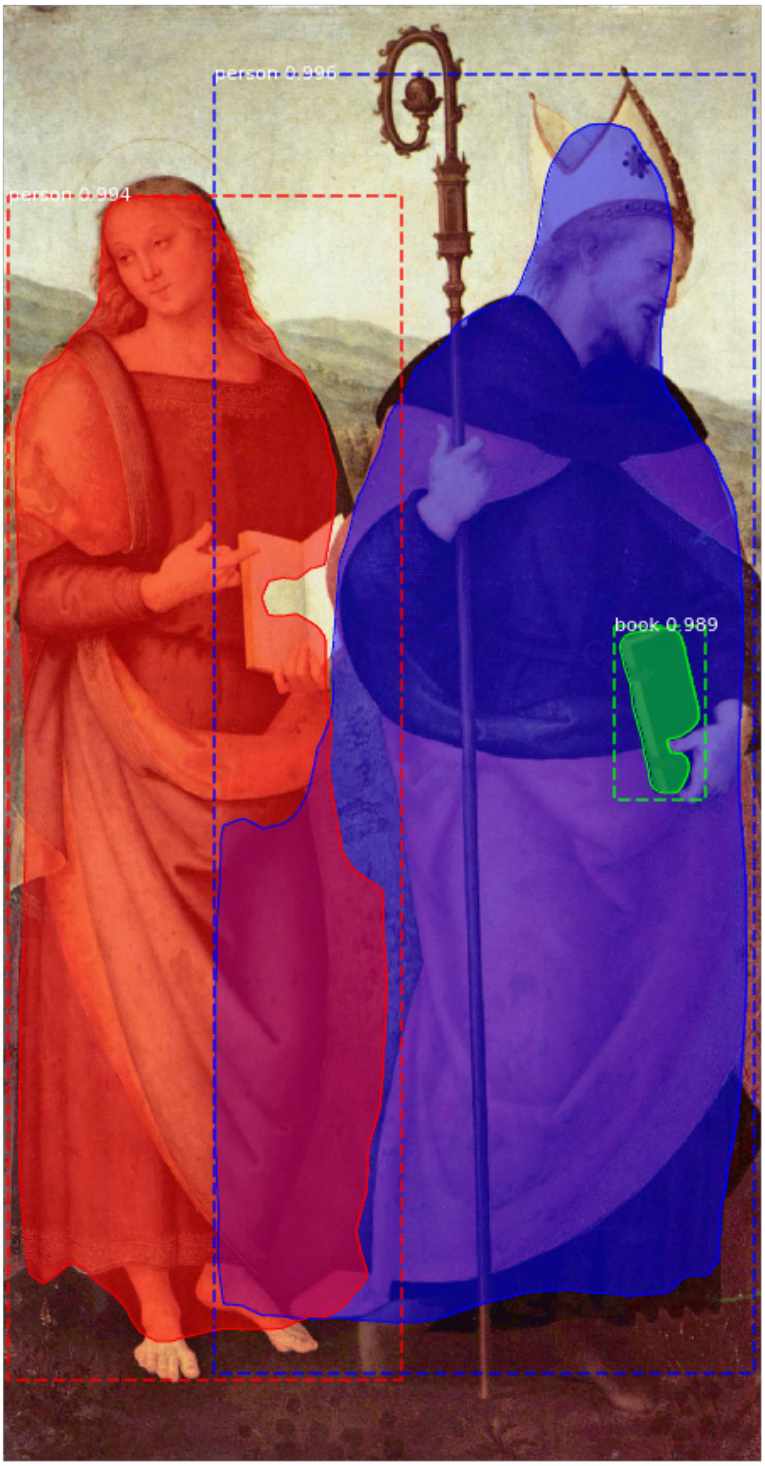
In [4]:
```
# COCO Class names
# Index of the class in the list is its ID. For example, to get ID of
# the teddy bear class, use: class_names.index('teddy bear')
classes = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',
               'bus', 'train', 'truck', 'boat', 'traffic light',
               'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird',
               'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
               'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie',
               'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
               'kite', 'baseball bat', 'baseball glove', 'skateboard',
               'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup',
               'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
               'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
               'doughnut', 'cake', 'chair', 'couch', 'pot plant', 'bed',
               'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote',
               'keyboard', 'cell phone', 'microwave', 'oven', 'toaster',
               'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors',
               'teddy bear', 'hair drier', 'toothbrush']
```

In [11]:
```
import pandas as pd
file_names = next(os.walk(IMAGE_DIR))[2]
image = skimage.io.imread(os.path.join(IMAGE_DIR, random.choice(file_names)))
century=20
# Run detection
results = model.detect([image],verbose=0)[0]
# Visualize results
visualize.display_instances(image, results['rois'], results['masks'], results
['class_ids'],
                            classes, results['scores'])
```

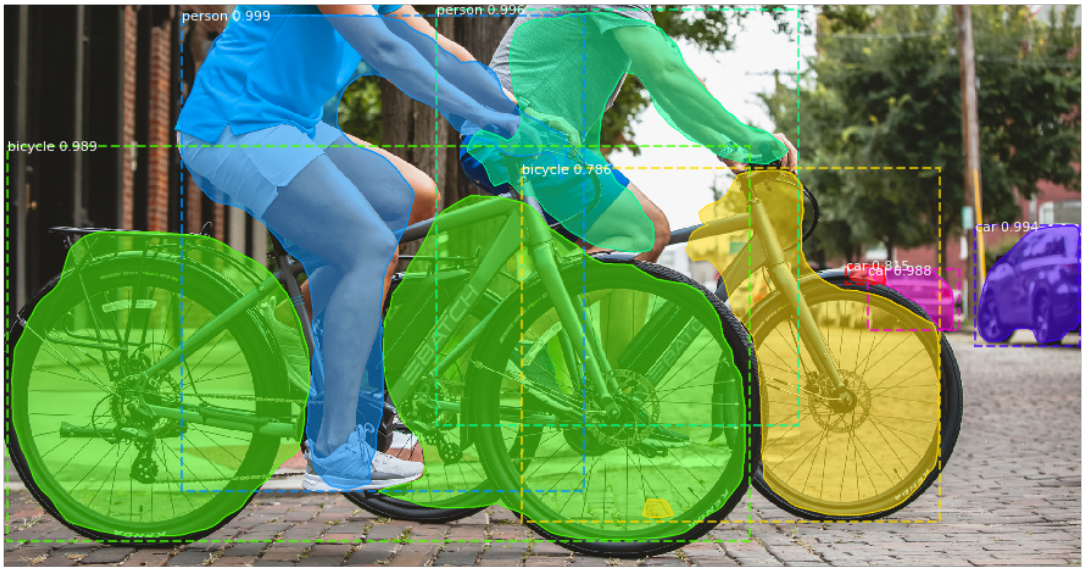


In [14]:
```
century=13
res_time=model.time_matrix_application(image,century,classes)
visualize.display_instances(image, res_time['rois'], res_time['masks'], res_ti
me['class_ids'],
                            classes, res_time['scores'])
```

```
probs                    shape: (1, 1000, 81)         min:    0.00000  max:
1.00000  float32
INFO: Time-matrix loaded correctly
INFO: Class  person . No need to apply time-matrix
INFO: Class  person . No need to apply time-matrix
INFO: Matrix of possible classes for class  cell phone
     Name      Prob
2  remote  0.004145
1  bottle  0.003263
0    book  0.001450
INFO: For class  cell phone  was chosen class  book
```



In [17]:
```
file_names = next(os.walk(IMAGE_DIR))[2]
image = skimage.io.imread(os.path.join(IMAGE_DIR, random.choice(file_names)))
century=20
# Run detection
results = model.detect([image],verbose=0)[0]
# Visualize results
visualize.display_instances(image, results['rois'], results['masks'], results
['class_ids'],
                            classes, results['scores'])
```



In [20]:
```
century=14
res_time=model.time_matrix_application(image,century,classes)
visualize.display_instances(image, res_time['rois'], res_time['masks'], res_ti
me['class_ids'],
                            classes, res_time['scores'])
```

```
probs                    shape: (1, 1000, 81)         min:    0.00000  max:
0.99999  float32
INFO: Time-matrix loaded correctly
INFO: Class  person . No need to apply time-matrix
INFO: Class  person . No need to apply time-matrix
INFO: Matrix of possible classes for class  car
         Name      Prob
3       truck  0.012431
1  motorcycle  0.003825
2      person  0.000885
0         bus  0.000112
INFO: For class  car  was chosen class  truck
INFO: Matrix of possible classes for class  bicycle
         Name      Prob
3  motorcycle  0.021593
2       horse  0.010329
0       bench  0.009906
1       chair  0.004155
INFO: For class  bicycle  was chosen class  horse
INFO: Matrix of possible classes for class  car
         Name      Prob
3       truck  0.012431
1  motorcycle  0.003825
2      person  0.000885
0         bus  0.000112
INFO: For class  car  was chosen class  truck
INFO: Matrix of possible classes for class  car
         Name      Prob
1  motorcycle  0.003825
2      person  0.000885
0         bus  0.000112
INFO: For class  car  was chosen class  person
INFO: Matrix of possible classes for class  bicycle
         Name      Prob
2  motorcycle  0.021593
0       bench  0.009906
1       chair  0.004155
INFO: For class  bicycle  was chosen class  bench
```

In [168]:
```
from shapely.geometry import Polygon
import heapq
def overlap_detection(rect1,rect2):
    try:
        p1 = Polygon([(rect1[0],rect1[1]), (rect1[1],rect1[1]),(rect1[2],rect1
[3]),(rect1[2],rect1[1])])
        p2 = Polygon([(rect2[0],rect2[1]), (rect2[1],rect2[1]),(rect2[2],rect2
[3]),(rect2[2],rect2[1])])
        return(p1.intersects(p2))
    except:
        return True

def time_matrix_application(img,century):
    detection = model.detect([image],verbose=0)
    # Visualize results
    r = detection[0]
    mrcnn = model.run_graph([img], [("probs", model.keras_model.get_layer("mrc
nn_class").output)])
    time_matrix=pd.read_csv("time_matrix.csv")
    print("INFO: Time-matrix loaded correctly")
    final_prob=pd.DataFrame()
    for i in mrcnn['probs'][0]:
        prob=pd.DataFrame()
        prob_data=[]
        for j in i:
            prob_data.append("{:.12f}".format(float(j)))
        prob["Name"]=class_names
        prob["Prob"]=prob_data
        final_prob=pd.concat([final_prob,prob],axis=0, ignore_index=True)
    final_prob["Prob"]=final_prob["Prob"].astype(float)

    for object_id in range(0,len(r["class_ids"])):
        object_time=time_matrix[time_matrix["Item"]==class_names[r["class_id
s"][object_id]]]
        #print(object_time)
        if object_time.reset_index().loc[0,"Cent"]>century:
            d_object=class_names[r["class_ids"][object_id]]
            middle_prop=pd.DataFrame(columns=["Name","Prob"])
            for i in range(0,len(final_prob),81):
                probs=[]
                for j in range(i,i+81):
                    # print(prob.loc[j,"Prob"])
                    probs.append(final_prob.loc[j,"Prob"])
                #print(probs)
                items=heapq.nlargest(4, range(len(probs)), key=probs.__getitem
__)
                if class_names[items[0]]==d_object:
                    for it in range(1,len(items)):
                        if items[it] not in r["class_ids"]:
                            middle_prop.loc[len(middle_prop)]=[class_names[ite
ms[it]], probs[items[it]]]
                        else:
                            position=np.where(r["class_ids"]==items[it])[0]
                            #print(position)
                            overlap_flag=False
                            for pos in position:
                                if overlap_detection(r["rois"][pos],r["roi
s"][object_id])==True:
                                    overlap_flag=True
                                    #print(pos)
                            if overlap_flag==False:
                                middle_prop.loc[len(middle_prop)]=[class_names
[items[it]], probs[items[it]]]
            middle_prop=middle_prop[middle_prop["Name"]!="BG"]
            print("INFO: Matrix of possible classes for class ",d_object)
            print(middle_prop.groupby("Name", as_index=False).mean().sort_valu
es(by=["Prob"],ascending=False))
            position_obj=0
```