In [22]:

```
# TensorFlow Libraries
import tensorflow as tf
import tensorflow_hub as hub

# For downloading the image.
import matplotlib.pyplot as plt
import tempfile
from six.moves.urllib.request import urlopen
from six import BytesIO

# For drawing onto the image.
import numpy as np
from PIL import Image
from PIL import ImageColor
from PIL import ImageDraw
from PIL import ImageFont
from PIL import ImageOps

# For measuring the inference time.
import time

# Generic
import os
import random
```

In [41]:

```
random_images = 'D:/DeePlearning/DATASET/ojectdetection/images'
```

In [24]:

```
# Function to Display Image
def display_image(image,file):
    fig = plt.figure(figsize=(15, 10))
    plt.grid(False)
    plt.imshow(image)
    plt.title(file, fontsize=20)


# Function to Resize Image
def resize_image(img, new_width=300, new_height=300,display=False):
    img_path = os.path.join(random_images,img)
    pil_image = Image.open(img_path)
    pil_image = ImageOps.fit(pil_image, (new_width, new_height), Image.ANTIALIAS)
    pil_image_rgb = pil_image.convert("RGB")
    if display:
        display_image(pil_image, img)
```

In [25]:

```
# Applying the Function
resize_image(random.choice(os.listdir(random_images)),400,200,True)
```

In [26]:

```
# Function to Draw Bounding Boxes on Image
def draw_bounding_box_on_image(image,ymin,xmin,ymax,xmax,color,font,thickness=4,display_str

    # Adds a bounding box to an image.
    draw = ImageDraw.Draw(image)
    im_width, im_height = image.size
    (left, right, top, bottom) = (xmin * im_width, xmax * im_width,ymin * im_height, ymax *
    draw.line([(left, top), (left, bottom), (right, bottom), (right, top),(left, top)],widt

    # If the total height of the display strings added to the top of the bounding
    # box exceeds the top of the image, stack the strings below the bounding box
    # instead of above.
    display_str_heights = [font.getsize(ds)[1] for ds in display_str_list]

    # Each display_str has a top and bottom margin of 0.05x.
    total_display_str_height = (1 + 2 * 0.05) * sum(display_str_heights)

    if top > total_display_str_height:
        text_bottom = top
    else:
        text_bottom = top + total_display_str_height

    # Reverse list and print from bottom to top.
    for display_str in display_str_list[::-1]:
        text_width, text_height = font.getsize(display_str)
        margin = np.ceil(0.05 * text_height)
        draw.rectangle([(left, text_bottom - text_height - 2 * margin),(left + text_width,
        draw.text((left + margin, text_bottom - text_height - margin),display_str, fill="bl
        text_bottom -= text_height - 2 * margin


# Function to Draw Boxes
def draw_boxes(image, boxes, class_names, scores, max_boxes=3, min_score=0.1):

    # Overlay labeled boxes on an image with formatted scores and label names.
    colors = list(ImageColor.colormap.values())
    font = ImageFont.load_default()

    for i in range(min(boxes.shape[0], max_boxes)):
        if scores[i] >= min_score:
            ymin, xmin, ymax, xmax = tuple(boxes[i])
            display_str = "{}: {}%".format(class_names[i].decode("ascii"),int(100 * scores[
            color = colors[hash(class_names[i]) % len(colors)]
            image_pil = Image.fromarray(np.uint8(image)).convert("RGB")

            draw_bounding_box_on_image(image_pil,ymin,xmin,ymax,xmax,color,font,display_str
            np.copyto(image, np.array(image_pil))
    return image
```

In [27]:

```
# FasterRCNN + InceptionResNet v2 Path
model_path = "https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1"

# Define Model
PreTrained_Model = hub.load(model_path).signatures['default']
```

```
INFO:tensorflow:Saver not created because there are no variables in the grap
h to restore

INFO:tensorflow:Saver not created because there are no variables in the grap
h to restore
```

In [28]:

```
# Function to Load Image from Random Image Folder using TensorFlow
def load_img(img_file):
    path = os.path.join(random_images,img_file)
    img = tf.io.read_file(path)
    img = tf.image.decode_jpeg(img, channels=3)
    return img
```

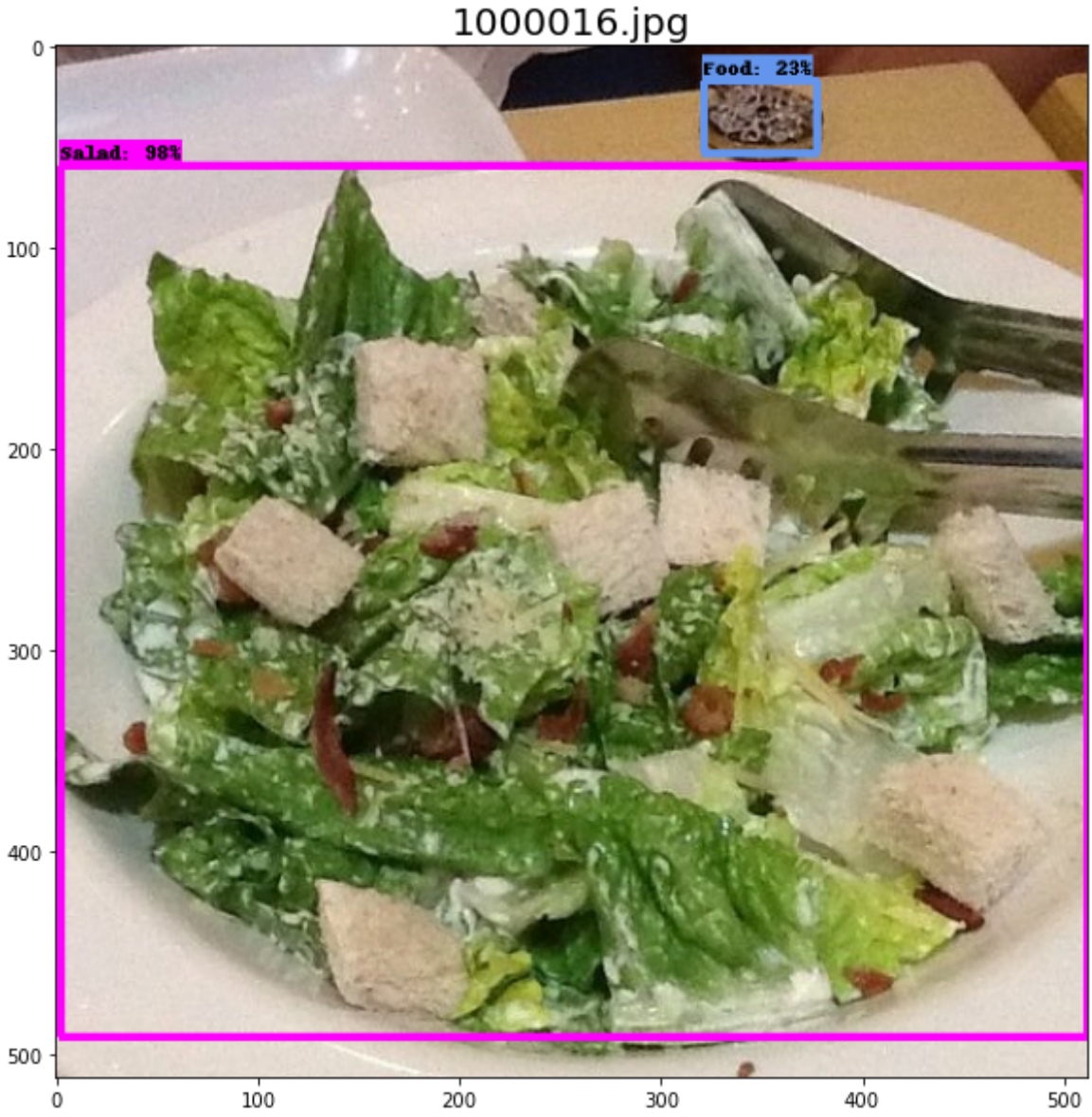
In [29]:

```
# Function to Detect Object using TensorFlow's Pretrained Model
def detect(model, img_file):

    img = load_img(img_file)

    converted_img  = tf.image.convert_image_dtype(img, tf.float32)[tf.newaxis, ...]

    start_time = time.time()   # Start Timer

    result = model(converted_img)   # Detection Using Pretrained Model downloaded above

    end_time = time.time()    # End Timer

    result = {key:value.numpy() for key,value in result.items()}

    print("Found %d objects." % len(result["detection_scores"]))
    print("Inference time (secs): ",'{:.2}'.format(end_time-start_time))

    image_with_boxes = draw_boxes(img.numpy(), result["detection_boxes"],result["detection_

    display_image(image_with_boxes,img_file)
```

In [36]:

```
# Running the Detect Function
detect(PreTrained_Model, random.choice(os.listdir(random_images)))
```

```
Found 100 objects.
Inference time (secs):  7.5e+01
```



In [31]:

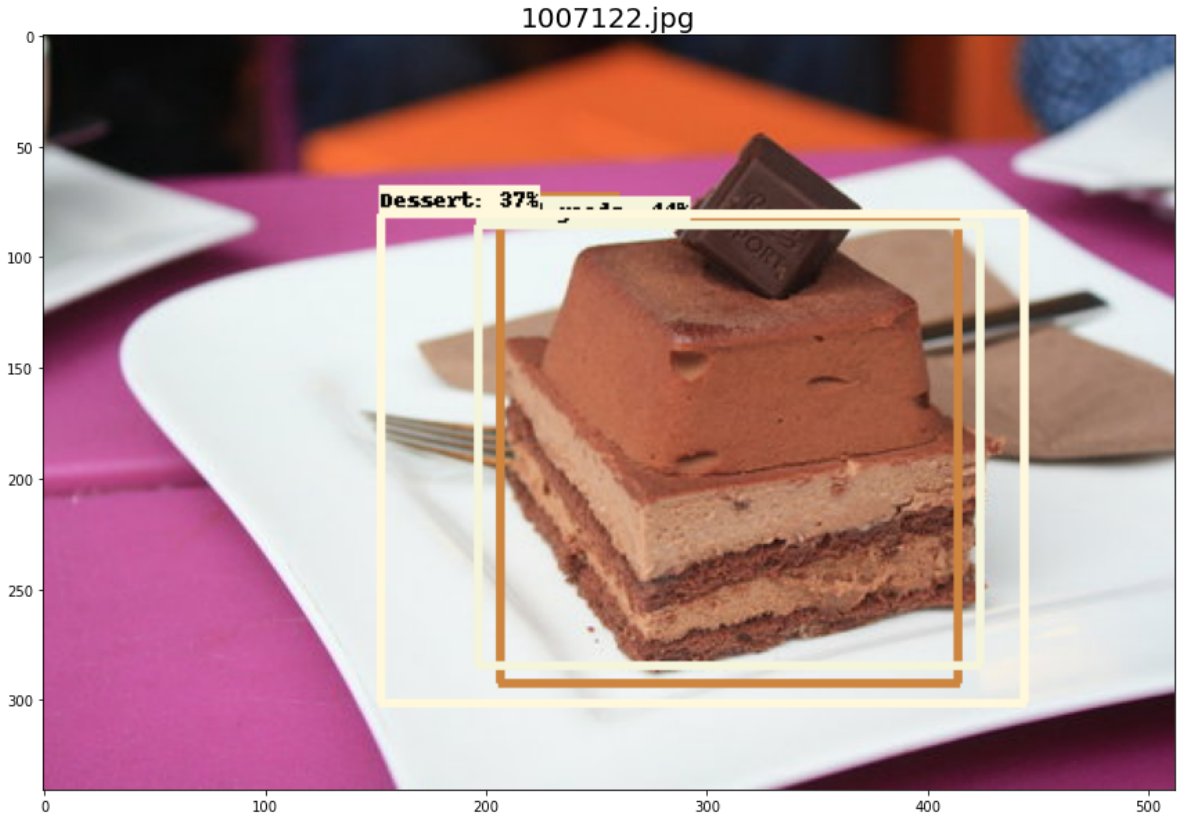
```
# Running the Detect Function
detect(PreTrained_Model, random.choice(os.listdir(random_images)))
```

```
Found 100 objects.
Inference time (secs):  8.8e+01
```

In [37]:

```
detect(PreTrained_Model, random.choice(os.listdir(random_images)))
```

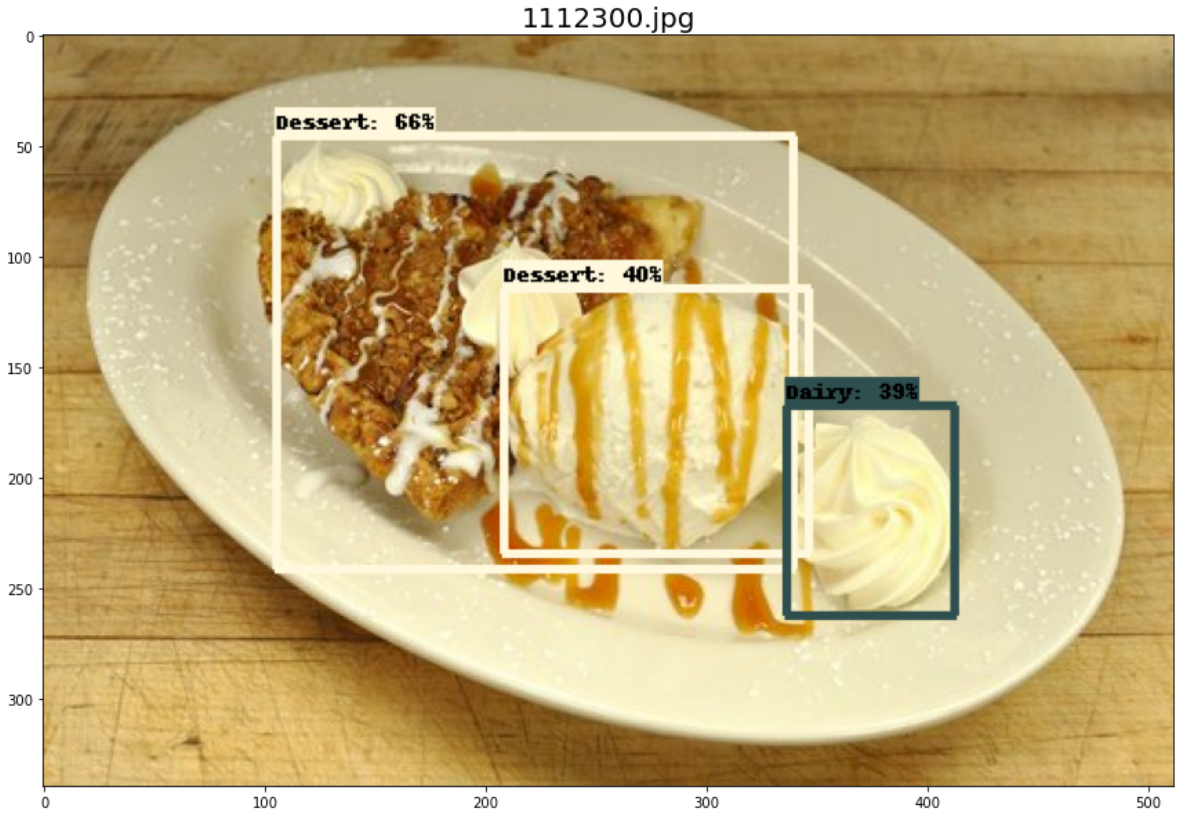
```
Found 100 objects.
Inference time (secs):  9.3e+01
```



In [42]:

```
detect(PreTrained_Model, random.choice(os.listdir(random_images)))
```

```
Found 100 objects.
Inference time (secs):  8.7e+01
```



In [45]:

```
detect(PreTrained_Model, random.choice(os.listdir(random_images)))
```

```
Found 100 objects.
Inference time (secs):  7.4e+01
```