1) **Importing Libraries:**

```
import cv2 # image processing, object detection
import numpy as np # blob creation, mathematical operations
import matplotlib.pyplot as plt #displaying images
import math #calculations for surface area
```

- "cv2" for image processing and object detection.
- "numpy" for array manipulations and mathematical operations.
- "matplotlib.pyplot" for displaying images.
- "math" for calculating surface area.

2) **Load image:**

```
img = cv2.imread("mobile and cup.jpg")  # Replace with your image path
height, width, channels = img.shape
plt.imshow(img)
plt.show()
```

- Loads the image from its source using OpenCV.
- Retrieves the dimensions of the image (height, width, and number of channels).
- Displays the loaded image using Matplotlib.

3) **Load YOLO or other preferred object detection model:**

```
net = cv2.dnn.readNet("yolov4.weights", "yolov4.cfg") # download these files from OpenCV official github repo
classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]
```

- Loads the YOLOv4 model using its weights and configuration files.
- Reads the class names from the "coco.names" file into a list called "classes".
- Retrieves the names of all the layers in the network.
- Identifies the output layers of the network, which are needed for object detection.

4) **Resize image and perform object detection:**

blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(output_layers)

- Preprocesses the image by creating a blob from the image, which resizes and normalizes it.
- Sets the blob as the input to the network.
- Performs a forward pass through the network to get the output from the output layers.

5) **Initialize variables for object detection:**

class_ids = []

confidences = []

boxes = []

- "class_ids": the class ID of the detected objects.
- "confidences": the confidence scores of the detected objects.
- "boxes": the bounding boxes of the detected objects.

6) **Function to calculate surface area based on shape:**

```
def calculate_surface_area(shape, w, h):
    if shape == 'rectangle':
        return w * h
    elif shape == 'circle':
        radius = w / 2
        return math.pi * (radius ** 2)
    else:
        return None
```

- Calculates the surface area based on the shape of the object.
- Returns the area for rectangles and circles. If the shape is not recognized, it returns "None".

7) **Detect and classify shapes for each object:**

```
for out in outs:
    for detection in out:
        scores = detection[5:]
```

```
class_id = np.argmax(scores)
confidence = scores[class_id]
if confidence > 0.5:  # Confidence threshold
    # Object detected
    center_x = int(detection[0] * width)
    center_y = int(detection[1] * height)
    w = int(detection[2] * width)
    h = int(detection[3] * height)

    # Rectangle coordinates
    x = int(center_x - w / 2)
    y = int(center_y - h / 2)

    boxes.append([x, y, w, h])
    confidences.append(float(confidence))
    class_ids.append(class_id)
```

- Iterates over the network outputs.
- Extracts the scores for each class and finds the class with the highest score.
- Checks if the confidence score is above a threshold (0.5).
- Calculates the center, width, and height of the bounding box.
- Converts the center coordinates to the top-left coordinates.
- Appends the bounding box coordinates, confidence score, and class ID to their respective lists.

8) **Non-max suppression:**

```
indexes = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5, nms_threshold=0.4)
```

- Remove overlapping bounding boxes based on their confidence scores.
- Retain only the most accurate bounding boxes.

9) **Detect and calculate surface area for each object:**

```
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
```

```
        label = str(classes[class_ids[i]])
# Assume flat object for simplicity
        if w / h < 1:
            shape = 'rectangle'  # Assume the detected object is rectangular
        elif w/h > 1:
            shape = 'circle'     # Assume the detected object is circular
        surface_area = calculate_surface_area(shape, w, h)
        print(f"Detected Object: {label}, Shape: {shape}, Surface Area: {surface_area:.2f} square units")
# Draw bounding box and label on the image
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(img, f"{label} {shape} {surface_area:.2f}", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)
```

- Iterates over the bounding boxes.
- Checks if the bounding box index is in the retained list from non-max suppression.
- Retrieves the bounding box coordinates, width, and height.
- Assumes the shape of the object based on the aspect ratio (rectangular or circular).
- Calculates the surface area using the “calculate_surface_area” function.
- Prints the detected object's label, shape, and surface area.
- Draws the bounding box and label on the image.

10) Convert image to RGB (Matplotlib expects RGB format):

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Display using Matplotlib
plt.figure(figsize=(10, 8))
plt.imshow(img_rgb)
plt.title('Object Detection and Surface Area Calculation')
plt.axis('off')
plt.show()
```

- Converts the image from BGR to RGB format, as Matplotlib expects RGB format.
- Displays the processed image with bounding boxes and labels using Matplotlib.

This entire code effectively detects objects in an image, classifies their shapes, calculates their surface areas, and displays the results.

❖ **Concepts Used in the Code**

1. Image Preprocessing

Before feeding an image to a neural network, it needs to be preprocessed to match the input requirements of the network. This typically involves resizing, normalization, and channel swapping.

- **Resizing**: The image is resized to a fixed dimension (416x416) to match the input size expected by the YOLO model.
- **Normalization**: Pixel values are scaled to the range [0, 1] using a scaling factor (0.00392).
- **Channel Swapping**: OpenCV uses BGR format by default, while most deep learning models expect RGB format. The blobFromImage function handles this conversion if specified.

2. Object Detection with YOLO

YOLO (You Only Look Once) is a real-time object detection system that detects objects within an image in a single forward pass through the network.

- **YOLOv4**: This is a version of YOLO with improvements in accuracy and speed. It uses a single neural network to predict bounding boxes and class probabilities directly from full images.
- **Forward Pass**: The processed image blob is passed through the network to get detection predictions.

3. Bounding Box Calculations

YOLO outputs bounding boxes in a format relative to the image dimensions. These need to be converted to absolute coordinates to draw boxes on the image.

- **Center Coordinates**: center_x and center_y are the center coordinates of the bounding box.
- **Width and Height**: w and h are the width and height of the bounding box.
- **Top-left Corner**: The top-left corner (x, y) is calculated from the center coordinates.

4. Non-Max Suppression (NMS)

NMS is a technique used to eliminate redundant overlapping bounding boxes for the same object. It ensures that only the bounding box with the highest confidence is retained.

- **Score Threshold**: Filters out detections with low confidence scores.
- **NMS Threshold**: Determines the threshold for considering two bounding boxes as overlapping.

Example of NMS: Assume there are two overlapping boxes for a detected object with confidence scores 0.9 and 0.8. NMS will keep the box with 0.9 confidence and discard the other.

5. Shape Detection and Surface Area Calculation

To determine the surface area of detected objects, the shape of each object needs to be classified (simplified as rectangles and circles here).

- **Shape Classification**: Based on the aspect ratio (width-to-height), objects are classified as rectangles or circles.
- **Surface Area Calculation**: Uses basic geometric formulas.

Example:

- For a rectangle with width 10 and height 5, the area is 10×5=5010 \times 5 = 5010×5=50.
- For a circle with diameter 10 (width), the radius is 555, and the area is π×52≈78.54 \pi \times 5^2 \approx 78.54π×52≈78.54.

6. Deep Learning Framework (OpenCV DNN)

OpenCV's Deep Neural Network (DNN) module allows using pre-trained models for tasks such as object detection, image classification, and face recognition. The cv2.dnn.readNet function is used to load a pre-trained model.

7. Drawing on Images

OpenCV provides functions to draw shapes and text on images, which is useful for visualizing detection results.

- **cv2.rectangle**: Draws rectangles (bounding boxes).
- **cv2.putText**: Draws text labels.