1. In the below given cell, shape of the boxes.eval() is (1783,4). Why are there 1783 boxes?

   The number of boxes is influenced by the random initialization of box_confidence, boxes, and box_class_probs. The threshold applied in yolo_filter_boxes filters out boxes with low confidence scores, retaining only the boxes with scores above the threshold. The specific value 1783 comes from how many boxes passed this filtering.

2. Explain the reason for it. What is the maximum number and minimum number you can get for that? Write these answers in a word file.

   Minimum: Theoretically, the minimum number of boxes can be 0 if no boxes pass the confidence threshold.

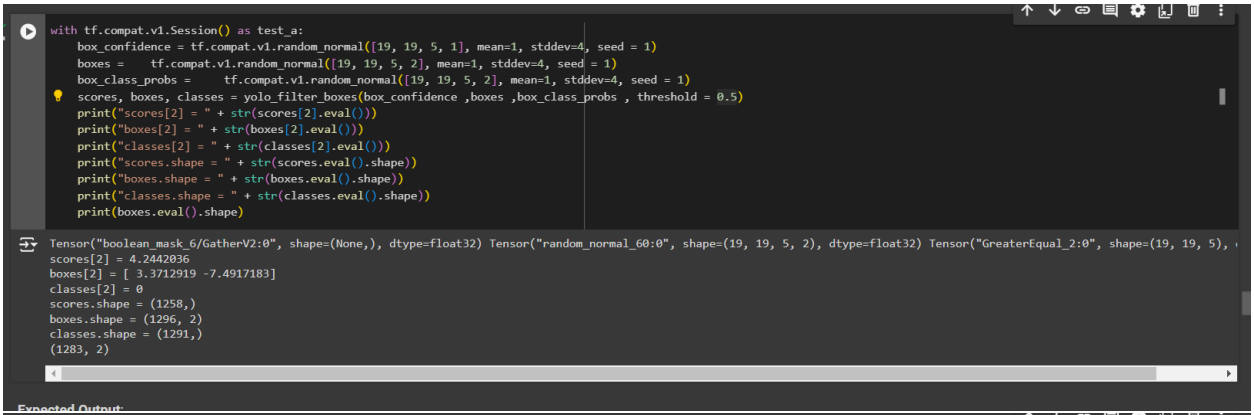
   Maximum: The maximum number of boxes is limited by the grid size and the number of anchor boxes. Given a 19x19 grid and 5 anchors, the maximum number would be 19 * 19 * 5 = 1805.

   - Change the values like mean and stddev in lines 2 and 4 as well as threshold value in line 5 and observe the different values you get for the boxes.eval().shape.

     Changing the values of `mean` and `stddev` affects the distribution of random values, altering the confidence scores and thus the number of boxes that pass the threshold. Similarly, adjusting the `threshold` directly impacts the number of boxes retained after filtering.

```
1 with tf.compat.v1.Session() as test_a:
2     box_confidence = tf.compat.v1.random_normal([19, 19, 5, 1], mean=1, stddev=4, seed = 1)
3     boxes = tf.compat.v1.random_normal([19, 19, 5, 4], mean=1, stddev=4, seed = 1)
4     box_class_probs = tf.compat.v1.random_normal([19, 19, 5, 80], mean=1, stddev=4, seed = 1)
5     scores, boxes, classes = yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold = 0.5)
6     print("scores[2] = " + str(scores[2].eval()))
7     print("boxes[2] = " + str(boxes[2].eval()))
8     print("classes[2] = " + str(classes[2].eval()))
9     print("scores.shape = " + str(scores.shape))
10    print("boxes.shape = " + str(boxes.shape))
11    print("classes.shape = " + str(classes.shape))
12    print(boxes.eval().shape)
```

```
Tensor("boolean_mask/GatherV2:0", shape=(None,), dtype=float32) Tensor("random_normal_1:0", shape=(19, 19, 5, 4
scores[2] = 10.750582
boxes[2] = [ 8.426533   3.2713668 -0.5313436 -4.9413733]
classes[2] = 7
scores.shape = (None,)
boxes.shape = (None, 4)
classes.shape = (None,)
(1783, 4)
```



```
with tf.compat.v1.Session() as test_a:
    box_confidence = tf.compat.v1.random_normal([19, 19, 5, 1], mean=1, stddev=4, seed = 1)
    boxes =    tf.compat.v1.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1)
    box_class_probs =     tf.compat.v1.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1)
    scores, boxes, classes = yolo_filter_boxes(box_confidence ,boxes ,box_class_probs , threshold = 0.5)
    print("scores[2] = " + str(scores[2].eval()))
    print("boxes[2] = " + str(boxes[2].eval()))
    print("classes[2] = " + str(classes[2].eval()))
    print("scores.shape = " + str(scores.eval().shape))
    print("boxes.shape = " + str(boxes.eval().shape))
    print("classes.shape = " + str(classes.eval().shape))
    print(boxes.eval().shape)

Tensor("boolean_mask_6/GatherV2:0", shape=(None,), dtype=float32) Tensor("random_normal_60:0", shape=(19, 19, 5, 2), dtype=float32) Tensor("GreaterEqual_2:0", shape=(19, 19, 5),
scores[2] = 4.2442036
boxes[2] = [ 3.3712919 -7.4917183]
classes[2] = 0
scores.shape = (1258,)
boxes.shape = (1296, 2)
classes.shape = (1291,)
(1283, 2)
```

Expected Output:

```
with tf.compat.v1.Session() as test_a:
    box_confidence = tf.compat.v1.random_normal([19, 19, 5, 1], mean=2, stddev=5, seed = 1)
    boxes =    tf.compat.v1.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1)
    box_class_probs =     tf.compat.v1.random_normal([19, 19, 5, 2], mean=2, stddev=6, seed = 1)
    scores, boxes, classes = yolo_filter_boxes(box_confidence ,boxes ,box_class_probs , threshold = 1)
    print("scores[2] = " + str(scores[2].eval()))
    print("boxes[2] = " + str(boxes[2].eval()))
    print("classes[2] = " + str(classes[2].eval()))
    print("scores.shape = " + str(scores.eval().shape))
    print("boxes.shape = " + str(boxes.eval().shape))
    print("classes.shape = " + str(classes.eval().shape))
    print(boxes.eval().shape)

Tensor("boolean_mask_9/GatherV2:0", shape=(None,), dtype=float32) Tensor("random_normal_63:0", shape=(19, 19, 5, 2), dtype=float32) Tensor("GreaterEqual_3:0", shape=(19, 19, 5),
scores[2] = 12.906715
boxes[2] = [ 3.3712919 -7.4917183]
classes[2] = 0
scores.shape = (1299,)
boxes.shape = (1316, 2)
classes.shape = (1325,)
(1315, 2)
```

```
with tf.compat.v1.Session() as test_a:
    box_confidence = tf.compat.v1.random_normal([19, 19, 5, 1], mean=4, stddev=6, seed = 1)
    boxes =    tf.compat.v1.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1)
    box_class_probs =     tf.compat.v1.random_normal([19, 19, 5, 2], mean=3, stddev=6, seed = 1)
    scores, boxes, classes = yolo_filter_boxes(box_confidence ,boxes ,box_class_probs , threshold = 2)
    print("scores[2] = " + str(scores[2].eval()))
    print("boxes[2] = " + str(boxes[2].eval()))
    print("classes[2] = " + str(classes[2].eval()))
    print("scores.shape = " + str(scores.eval().shape))
    print("boxes.shape = " + str(boxes.eval().shape))
    print("classes.shape = " + str(classes.eval().shape))
    print(boxes.eval().shape)

Tensor("boolean_mask_12/GatherV2:0", shape=(None,), dtype=float32) Tensor("random_normal_66:0", shape=(19, 19, 5, 2), dtype=float32) Tensor("GreaterEqual_4:0", shape=(19, 19, 5),
scores[2] = 103.765816
boxes[2] = [ 3.3712919 -7.4917183]
classes[2] = 0
scores.shape = (1333,)
boxes.shape = (1355, 2)
classes.shape = (1344,)
(1374, 2)
```

yolo_anchors.txt contains 10 values. They can be considered as height and width of 5 anchor boxes. What is the advantage of using such anchor boxes? Give the answers to these questions in the word file.

Anchor boxes allow YOLO to predict bounding boxes of different shapes and sizes efficiently. This is crucial for detecting objects of varying dimensions within the same image. By using predefined anchor boxes, the model can quickly converge during training and better handle objects of different aspect ratios.

What was the method used to determine the sizes of these anchor boxes?

The anchor box sizes are typically determined using k-means clustering on the dimensions of ground truth bounding boxes from the training dataset. The aim is to find a set

of anchor boxes that minimize the distance (often IoU-based) to the actual boxes in the dataset, ensuring the anchor boxes represent the most common object shapes.