(YOLO)

In [ ]:

```python
import cv2
import numpy as np
import tensorflow as tf
```

In [ ]:

```python
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

with open("coco.names", "r") as f: #coco.names has been modified for our lab inputs
    classes = [line.strip() for line in f.readlines()]

# Initialize the webcam
cap = cv2.VideoCapture(0)

while True:
    _, frame = cap.read()

    # Prepare the frame for YOLO
    height, width, channels = frame.shape
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    # Process the outputs
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # 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)

                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(frame, classes[class_id], (x, y + 30), cv2.FONT_HERSHEY_PLAIN, 1, (0, 25

    # Display the resulting frame
    cv2.imshow('Frame', frame)

    # Break the loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release everything if job is finished
cap.release()
cv2.destroyAllWindows()

```

(RCNN)

In [ ]:

```
!unzip Images.zip
```

```
  inflating: Images/airplane_305.jpg
  inflating: Images/airplane_304.jpg
  inflating: Images/airplane_303.jpg
  inflating: Images/airplane_292.jpg
  inflating: Images/airplane_291.jpg
  inflating: Images/airplane_290.jpg
  inflating: Images/airplane_289.jpg
  inflating: Images/airplane_288.jpg
  inflating: Images/airplane_287.jpg
  inflating: Images/airplane_286.jpg
  inflating: Images/airplane_138.jpg
  inflating: Images/airplane_137.jpg
  inflating: Images/airplane_136.jpg
  inflating: Images/airplane_135.jpg
  inflating: Images/airplane_134.jpg
  inflating: Images/airplane_133.jpg
  inflating: Images/airplane_132.jpg
  inflating: Images/airplane_131.jpg
  inflating: Images/airplane_130.jpg
  inflating: Images/airplane_129.jpg
  inflating: Images/airplane_128.jpg
```

In [ ]:

```
!unzip Airplanes_Annotations.zip
```

```
 inflating: Airplanes_Annotations/airplane_176.csv
extracting: Airplanes_Annotations/airplane_175.csv
extracting: Airplanes_Annotations/airplane_174.csv
 inflating: Airplanes_Annotations/airplane_173.csv
extracting: Airplanes_Annotations/airplane_172.csv
extracting: Airplanes_Annotations/airplane_171.csv
 inflating: Airplanes_Annotations/airplane_170.csv
 inflating: Airplanes_Annotations/airplane_169.csv
 inflating: Airplanes_Annotations/airplane_168.csv
 inflating: Airplanes_Annotations/airplane_167.csv
 inflating: Airplanes_Annotations/airplane_166.csv
 inflating: Airplanes_Annotations/airplane_165.csv
extracting: Airplanes_Annotations/airplane_164.csv
extracting: Airplanes_Annotations/airplane_163.csv
 inflating: Airplanes_Annotations/airplane_162.csv
extracting: Airplanes_Annotations/airplane_161.csv
 inflating: Airplanes_Annotations/airplane_160.csv
extracting: Airplanes_Annotations/airplane_159.csv
 inflating: Airplanes_Annotations/airplane_158.csv
extracting: Airplanes_Annotations/airplane_157.csv
```

In [ ]:

```python
import cv2
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from keras.models import Model
from keras.layers import Dense
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
```

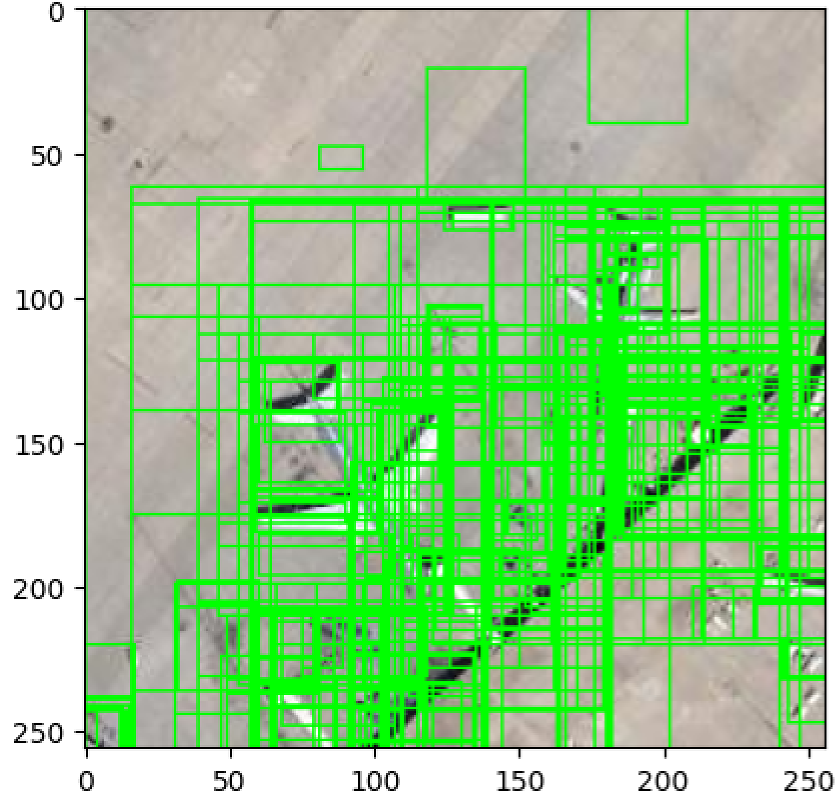
In [ ]:

```python
def load_image(file_path, file_name):
    return cv2.imread(os.path.join(file_path, file_name))

def calculate_iou(box1, box2):
    x_left = max(box1['x1'], box2['x1'])
    y_top = max(box1['y1'], box2['y1'])
    x_right = min(box1['x2'], box2['x2'])
    y_bottom = min(box1['y2'], box2['y2'])

    if x_right < x_left or y_bottom < y_top:
        return 0.0

    intersect_area = (x_right - x_left) * (y_bottom - y_top)
    box1_area = (box1['x2'] - box1['x1']) * (box1['y2'] - box1['y1'])
    box2_area = (box2['x2'] - box2['x1']) * (box2['y2'] - box2['y1'])

    iou = intersect_area / float(box1_area + box2_area - intersect_area)
    return iou

def custom_rcnn_model(input_shape):
    vgg = VGG16(weights='imagenet', include_top=True)
    for layer in vgg.layers[:15]:
        layer.trainable = False

    x = vgg.layers[-2].output
    predictions = Dense(2, activation="softmax")(x)
    model = Model(inputs=vgg.input, outputs=predictions)
    return model

def selective_search(image):
    ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
    ss.setBaseImage(image)
    ss.switchToSelectiveSearchFast()
    return ss.process()

from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import img_to_array

def preprocess_region(region):
    region = cv2.resize(region, (224, 224))
    region = img_to_array(region)
    region = preprocess_input(region)
    return region

def get_label_for_region(image_file, rect):
  return 1

```

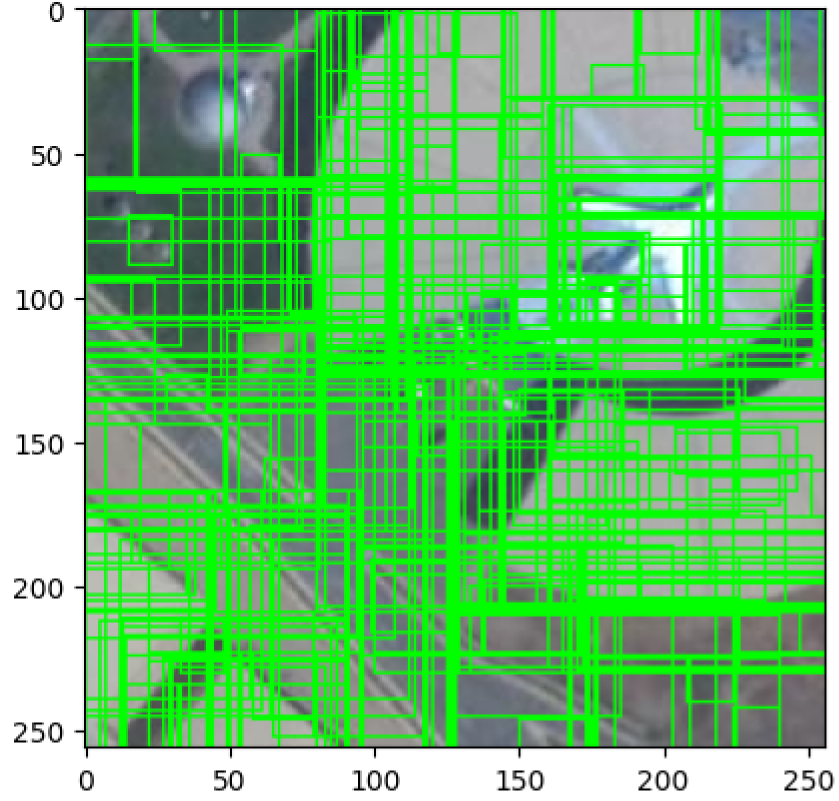
In [ ]:

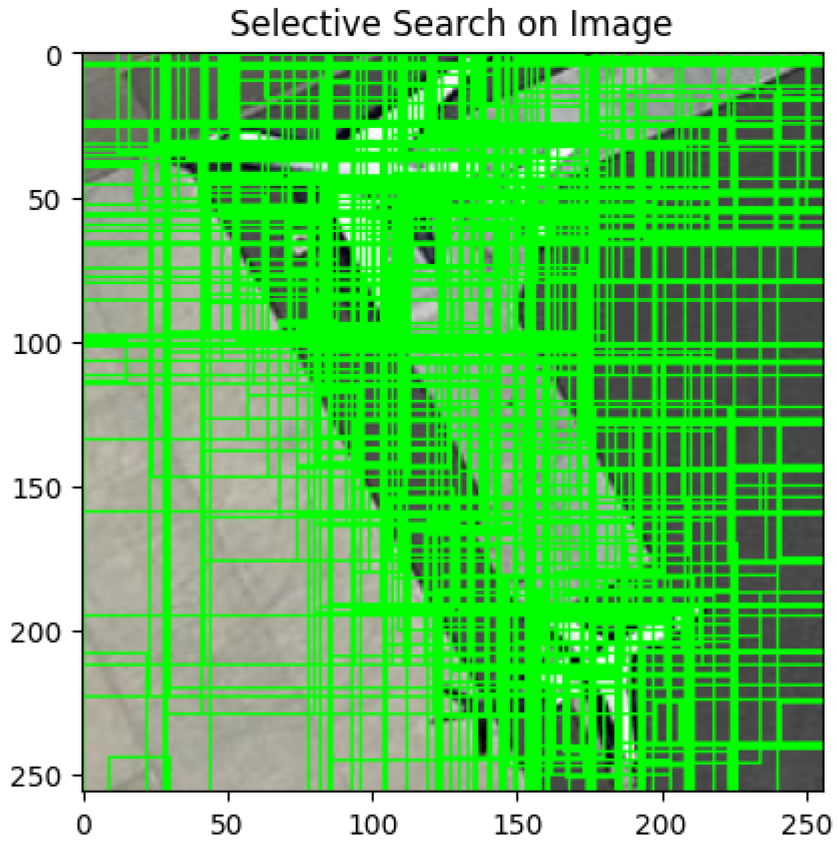
```python
image_dir = '/content/Images'
count = 0
for image_file in os.listdir(image_dir):
    if image_file.lower().endswith(('.png', '.jpg', '.jpeg')):
        image_path = os.path.join(image_dir, image_file)
        image = cv2.imread(image_path)

        rects = selective_search(image)
        image_out = image.copy()

        for i, rect in enumerate(rects):
            x, y, w, h = rect
            cv2.rectangle(image_out, (x, y), (x+w, y+h), (0, 255, 0), 1)

        image_out_rgb = cv2.cvtColor(image_out, cv2.COLOR_BGR2RGB)
        if(count < 3):
          count+=1
          plt.imshow(image_out_rgb)
          plt.title("Selective Search on Image")
          plt.show()
```

Selective Search on Image



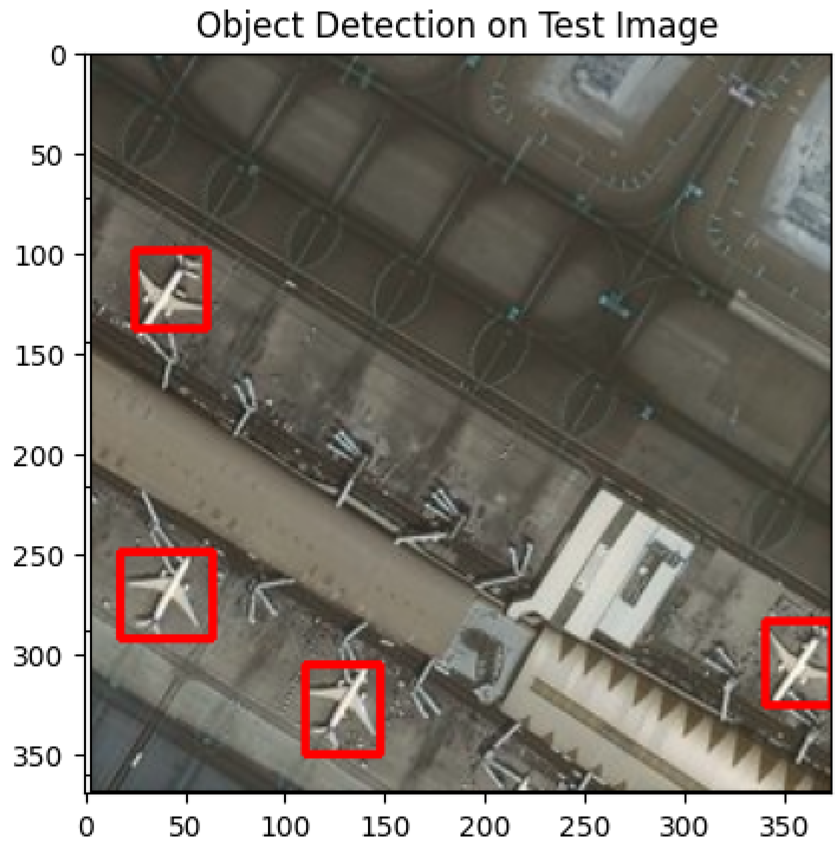
Selective Search on Image




Selective Search on Image
0
50
100
150
200
250
0
50
100
150
200
250

In [3]:

```python

train_images = []
train_labels = []

test_image = cv2.imread("/content/Images/428451.jpg")

for image_file in os.listdir(image_dir):
    if image_file.lower().endswith(('.png', '.jpg', '.jpeg')):
        image_path = os.path.join(image_dir, image_file)
        image = cv2.imread(image_path)

        rects = selective_search(image)

        for i, rect in enumerate(rects):
            x, y, w, h = rect
            region = image[y:y+h, x:x+w]

            processed_region = preprocess_region(region)

            label = get_label_for_region(image_file, rect)

            train_images.append(processed_region)
            train_labels.append(label)

train_images_np = np.array(train_images)
train_labels_np = np.array(train_labels)

model = custom_rcnn_model(train_images_np[0].shape)

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(train_images_np, train_labels_np, epochs=10, batch_size=32)

test_rects = selective_search(test_image)
test_image_out = test_image.copy()

for rect in test_rects:
    x, y, w, h = rect
    test_region = test_image[y:y+h, x:x+w]
    processed_test_region = preprocess_region(test_region)

    prediction = model.predict(np.array([processed_test_region]))
    if prediction[0][1] > 0.5:
        cv2.rectangle(test_image_out, (x, y), (x+w, y+h), (0, 255, 0), 1)

test_image_out_rgb = cv2.cvtColor(test_image_out, cv2.COLOR_BGR2RGB)
plt.imshow(test_image_out_rgb)
plt.title("Object Detection on Test Image")
plt.show()

```



In [ ]:

```python

```