Open Images Object Detection RVC 2020 edition

Detect objects in varied and complex images

Using FasterRCNN+InceptionResNet V2, an SSD-based object detection model trained on Open Images V4 with ImageNet pre-trained MobileNet V2 as image feature extractor.

+ Code + Markdown

[1]:
```
import pdb
import math
import itertools
import time
import gc
import pickle
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tempfile
import tensorflow as tf
import tensorflow_hub as hub

from PIL import Image, ImageColor, ImageDraw, ImageFont, ImageOps

import os

image_path = '/kaggle/input/open-images-object-detection-rvc-2020'
module_handle = 'https://tfhub.dev/google/faster_rcnn/openimages_v4/inception_resnet_v2/1'

print(tf.__version__)
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    # Restrict TensorFlow to only use the first GPU
    try:
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPU")
    except RuntimeError as e:
        # Visible devices must be set before GPUs have been initialized
        print(e)
```

```
2.1.0
1 Physical GPUs, 1 Logical GPU
```

Helper Functions

[2]:
```
# Resizes image to new_width x new_height and returns PIL file
def resize_image(path, new_width=256, new_height=256):
    _, filename = tempfile.mkstemp(suffix=".jpg")
    pil_image = Image.open(path)
    pil_image = ImageOps.fit(pil_image, (new_width, new_height), Image.ANTIALIAS)
    pil_image_rgb = pil_image.convert("RGB")
    pil_image_rgb.save(filename, format="JPEG", quality=90)
    # print('Resized image saved as: {}'.format(filename))
    return filename, pil_image
```

[3]:
```
# Display a PIL image file
def display_image(image):
    fig = plt.figure(figsize=(20, 15))
    plt.grid(False)
    plt.imshow(image)
```

[4]:
```
# Load image into TF
def load_img(path):
    print('loading image: {}'.format(path))
    img = tf.io.read_file(path)
    img = tf.image.decode_jpeg(img, channels=3)
    return img
```

[5]:
```
# Adds a bounding box to an image
def draw_bounding_box_on_image(image, ymin, xmin, ymax,
                               xmax, color, font,
                               thickness=4, display_str_list=()):
    draw = ImageDraw.Draw(image)
    im_width, im_height = image.size
```

```
    im_width, im_height = image.size
    (left, right, top, bottom) = (xmin * im_width, xmax * im_width,
                                  ymin * im_height, ymax * im_height)
    draw.line([(left, top), (left, bottom), (right, bottom), (right, top),
               (left, top)],
              width=thickness,
              fill=color)

    # 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, text_bottom)],
                       fill=color)
        draw.text((left + margin, text_bottom - text_height - margin),
                  display_str,
                  fill="black",
                  font=font)
        text_bottom -= text_height - 2 * margin
```

[6]:

```
# Draw object boxes for the images
def draw_boxes(image, boxes, class_names, scores, max_boxes=10, min_score=0.0):
    print('in draw boxes')
    colors = list(ImageColor.colormap.values())

    try:
        font = ImageFont.truetype("/usr/share/fonts/truetype/liberation/LiberationSansNarrow-Regular.ttf",25)
    except IOError:
        print("Font not found, using default font.")
        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[i]))
            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_list=[display_str])
        np.copyto(image, np.array(image_pil))
    return image
```

[7]:

```
# Run the detector on an image
def run_detector(detector, path, b=True):
    img = load_img(path)
    converted_img  = tf.image.convert_image_dtype(img, tf.float32)[tf.newaxis, ...]
    with tf.device('/GPU:0'):
        result = detector(converted_img)
        result = {key:value.numpy() for key,value in result.items()}
        image_with_boxes = None
        if b is True:
#             print(result)
            image_with_boxes = draw_boxes(
                img.numpy(), result["detection_boxes"],
                result["detection_class_entities"], result["detection_scores"])
        return image_with_boxes, result
```

[8]:

```
# Get the file name from the image id
def filename_from_id(id):
    return os.path.join(image_path, 'test/', '{}.jpg'.format(id) )
```

Resize and Display Sample Image

[9]:

```
sample_submission_df = pd.read_csv(f'{image_path}/sample_submission.csv')
image_ids = sample_submission_df['ImageId']
del sample_submission_df
```

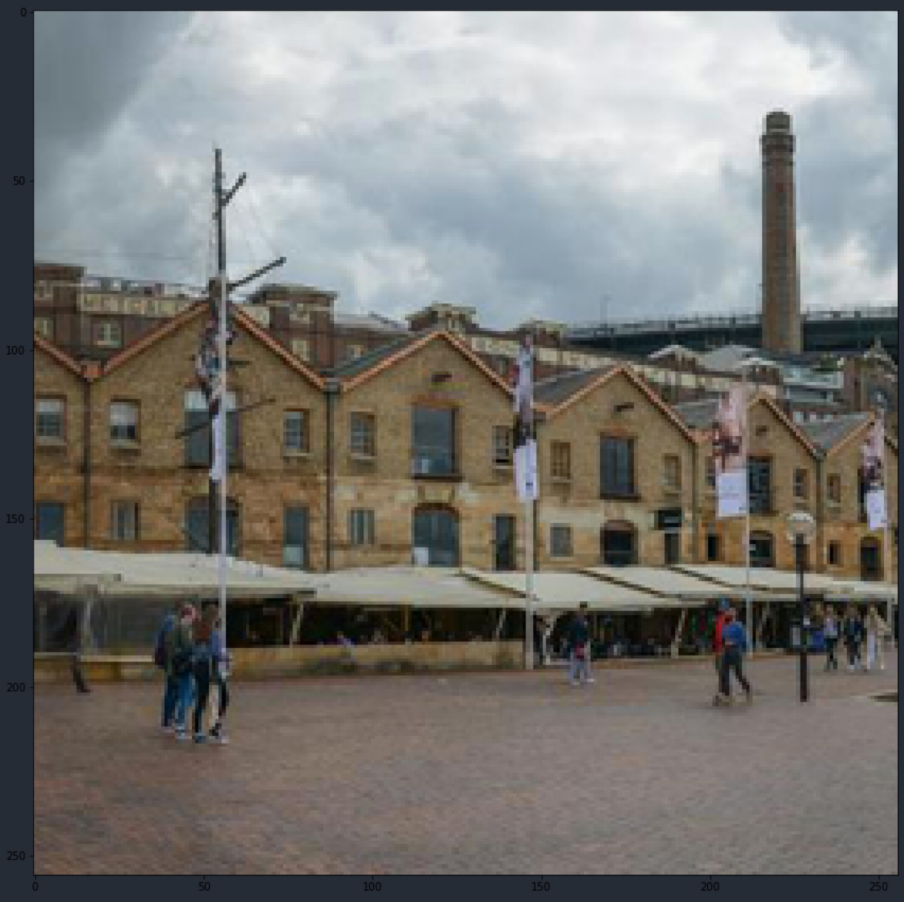
[32]:

```
test_img = 22
# Build a list of images

filename = filename_from_id(image_ids[test_img])

# Load, resize and display sample image
```

```
filename_r, pil_image = resize_image(filename)
display_image(pil_image)
del pil_image
```



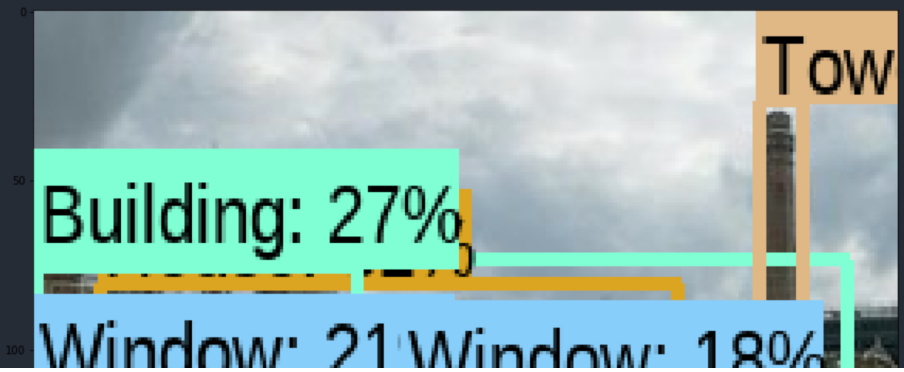
Run the Detector on a Single Image

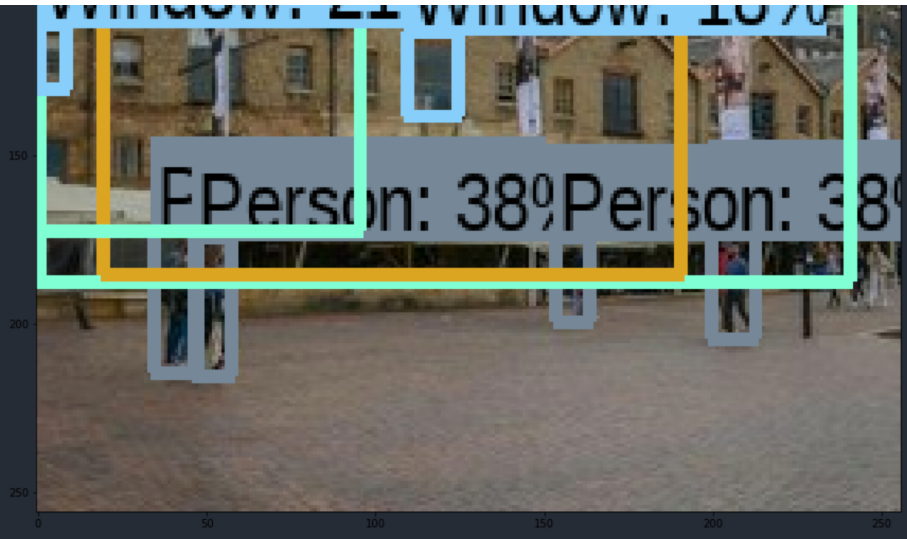
[34]:
```
print('Loading module...')
detector = hub.load(module_handle).signatures['default']
```

```
Loading module...
```

[35]:
```
print('Processing image: {}'.format(filename))
filename_r, pil_image = resize_image(filename)
image_with_boxes, result = run_detector(detector, filename_r)
print('Found {} objects'.format(len(result['detection_scores'])))
display_image(image_with_boxes)
del image_with_boxes
del pil_image
```

```
Processing image: /kaggle/input/open-images-object-detection-rvc-2020/test/effab8b7e832bf1c.jpg
loading image: /tmp/tmpoxzr0fox.jpg
in draw boxes
Found 100 objects
```





Create Submission

[13]:
```
def index_marks(nrows, chunk_size):
    return range(chunk_size, math.ceil(nrows / chunk_size) * chunk_size, chunk_size)
```

[14]:
```
def split(df, chunk_size):
    indices = index_marks(df.shape[0], chunk_size)
    return np.split(df, indices)
```

[15]:
```
# Create the prediction results string
def make_prediction_string(result, idx):
    class_name = result['detection_class_names'][idx].decode("utf-8")
    boxes = result['detection_boxes'][idx]
    score = result['detection_scores'][idx]
    return f"{class_name} {score} " + " ".join(map(str, boxes))
```

[16]:
```
# Formats the prediction results
def format_prediction_string(image_id, result):
    prediction_strings = [make_prediction_string(result, i) for i in range(len(result['detection_scores']))]
    return  " ".join(prediction_strings)
```

[17]:
```
def chunk_inference(chunk, detector, predictions):
    results = []
    for img in chunk:
        filename = filename_from_id(img)
        filename_r, pil_image = resize_image(filename)
        , result = run_detector(detector, filename_r, False)
        results.append(result)
    return results
```

[18]:
```
chunk_size = 2500
n_chunks = round(len(image_ids) / chunk_size)
chunks = split(image_ids, chunk_size)
print('Chunks: {}'.format(n_chunks))
```

```
Chunks: 40
```

[19]:
```
predictions = []
chunk_counter = 1
for c in chunks:
    print('Processing chunk {}'.format(chunk_counter))
    chunk_pred = chunk_inference(c, detector, predictions)
    predictions.append(chunk_pred)
    chunk_counter += 1
    del chunk_pred
del detector
```

```
Processing chunk 1
Processing chunk 2
Processing chunk 3
```

```
Processing chunk 4
Processing chunk 5
Processing chunk 6
Processing chunk 7
Processing chunk 8
Processing chunk 9
Processing chunk 10
Processing chunk 11
Processing chunk 12
Processing chunk 13
Processing chunk 14
Processing chunk 15
Processing chunk 16
Processing chunk 17
Processing chunk 18
Processing chunk 19
Processing chunk 20
Processing chunk 21
Processing chunk 22
Processing chunk 23
Processing chunk 24
Processing chunk 25
Processing chunk 26
Processing chunk 27
Processing chunk 28
Processing chunk 29
Processing chunk 30
Processing chunk 31
Processing chunk 32
Processing chunk 33
Processing chunk 34
Processing chunk 35
Processing chunk 36
Processing chunk 37
Processing chunk 38
Processing chunk 39
Processing chunk 40
```

[20]:
```
gc.collect()
```

Out[20]
```
1522155
```

[21]:
```
file = open('predictions_final.pkl', 'wb')
pickle.dump(predictions, file)
file.close()
```

[22]:
```
file = open('image_ids.pkl', 'wb')
pickle.dump(image_ids, file)
file.close()
```

[23]:
```
predictions = pickle.load(open( "predictions_final.pkl", "rb" ))
image_ids = pickle.load(open( "image_ids.pkl", "rb" ))
```

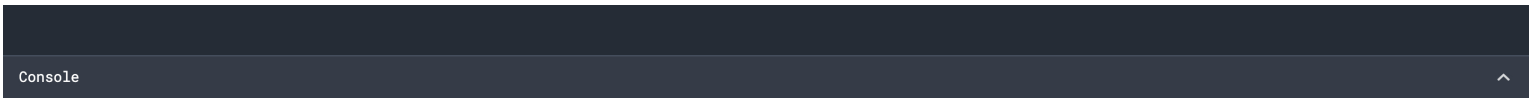
[24]:
```
preds_merged = list(itertools.chain.from_iterable(predictions))
```

[25]:
```
column_names = ['ImageID', 'RawPred','PredictionString']
output = pd.DataFrame(columns = column_names)
output['ImageID'] = image_ids
output['RawPred'] = preds_merged
for index, row in output.iterrows():
    row['PredictionString'] = format_prediction_string(row['ImageID'], row['RawPred'])
output.head()
```

Out[25]:

	ImageID	RawPred	PredictionString
0	b5d912e06f74e948	{'detection_class_labels': [313, 136, 447, 28,...	/m/050k8 0.4688078761100769 0.17766872 0.18811...
1	be137cf6bb0b62d5	{'detection_class_labels': [313, 136, 447, 28,...	/m/050k8 0.4688078761100769 0.17766872 0.18811...
2	8d65ca08cb5ce8e8	{'detection_class_labels': [313, 136, 447, 28,...	/m/050k8 0.4688078761100769 0.17766872 0.18811...
3	4d3ad1e52ad8c065	{'detection_class_labels': [313, 136, 447, 28,...	/m/050k8 0.4688078761100769 0.17766872 0.18811...
4	9b94408691c7d7bf	{'detection_class_labels': [313, 136, 447, 28,...	/m/050k8 0.4688078761100769 0.17766872 0.18811...

[26]:
```
output.drop('RawPred', axis=1, inplace=True)
output.to_csv('submission.csv', index=False)
```


Console