Images and masks splitting into multiple pieces in Python with Google Colab

A practical example of images and masks splitting into smaller parts

Oleksii Sheremet Aug 6, 2020 · 4 min read ★

Data labelers use special annotation tools for objects annotation. For example, the Computer Vision Annotation Tool (CVAT) is widely known in computer vision. Naturally, it is more convenient for labelers to work with high-resolution images. This is especially true when you need to mark a large number of objects.

In one of the roof segmentation tasks that I participated in, it was necessary to highlight triangular segments, quadrangular segments, other segments and edges of the roof. An example of such markup is shown in the following figure (white color for edges, red color for triangles, green color for quadrangles, blue color for other polygons):

Image is created by Oleksii Sheremet with matplotlib module

The original images were obtained from Google Earth at 2048x1208 pixels. The masks were annotated by data labelers using CVAT at the same resolution. To train the model, images and masks should be in a lower resolution (from 128x128 to 512x512 pixels). It is well known that image splitting is a technique most often used to slice a large image into smaller parts. Thus, the logical solution was to split the images and their corresponding masks into the parts with the same resolution.

All code for splitting was implemented in Google Colab. Let's take a closer look. Import libraries:

```
import os
import sys
import shutil
import glob
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
```

Mount the Google Drive (with images and masks) to Google Colab:

```
from google.colab import drive
drive.mount('/content/gdrive')
%cd "gdrive/My Drive/File Folder"
```

A useful function for creating a new directory and recursively deleting the contents of an existing one:

```
def dir_create(path):
    if (os.path.exists(path)) and (os.listdir(path) != []):
        shutil.rmtree(path)
        os.makedirs(path)
    if not os.path.exists(path):
        os.makedirs(path)
```

The crop function that goes over the original image are adjusted to the original image limit and contain the original pixels:

```
def crop(input_file, height, width):
    img = Image.open(input_file)
    img_width, img_height = img.size
    for i in range(img_height//height):
        for j in range(img_width//width):
            box = (j*width, i*height, (j+1)*width, (i+1)*height)
            yield img.crop(box)
```

The function for splitting images and masks into smaller parts (the height and width of the cropping window, and the starting number are taken as input parameters):

```
def split(inp_img_dir, inp_msk_dir, out_dir, height, width,
          start_num):
    image_dir = os.path.join(out_dir, 'images')
    mask_dir = os.path.join(out_dir, 'masks')
    dir_create(out_dir)
    dir_create(image_dir)
    dir_create(mask_dir)
    img_list = [f for f in
                os.listdir(inp_img_dir)
                if os.path.isfile(os.path.join(inp_img_dir, f))]
    file_num = 0
    for infile in img_list:
        infile_path = os.path.join(inp_img_dir, infile)
        for k, piece in enumerate(crop(infile_path,
                                       height, width), start_num):
            img = Image.new('RGB', (height, width), 255)
            img.paste(piece)
            img_path = os.path.join(image_dir,
                                    infile.split('.')[0]+ '_'
                                    + str(k).zfill(5) + '.png')
            img.save(img_path)
        infile_path = os.path.join(inp_msk_dir,
                                   infile.split('.')[0] + '.png')
        for k, piece in enumerate(crop(infile_path,
                                       height, width), start_num):
            msk = Image.new('RGB', (height, width), 255)
            msk.paste(piece)
            msk_path = os.path.join(mask_dir,
                                    infile.split('.')[0] + '_'
                                    + str(k).zfill(5) + '.png')
            msk.save(msk_path)
        file_num += 1
        sys.stdout.write("\rFile %s was processed." % file_num)
        sys.stdout.flush()
```

Let’s set the necessary variables:

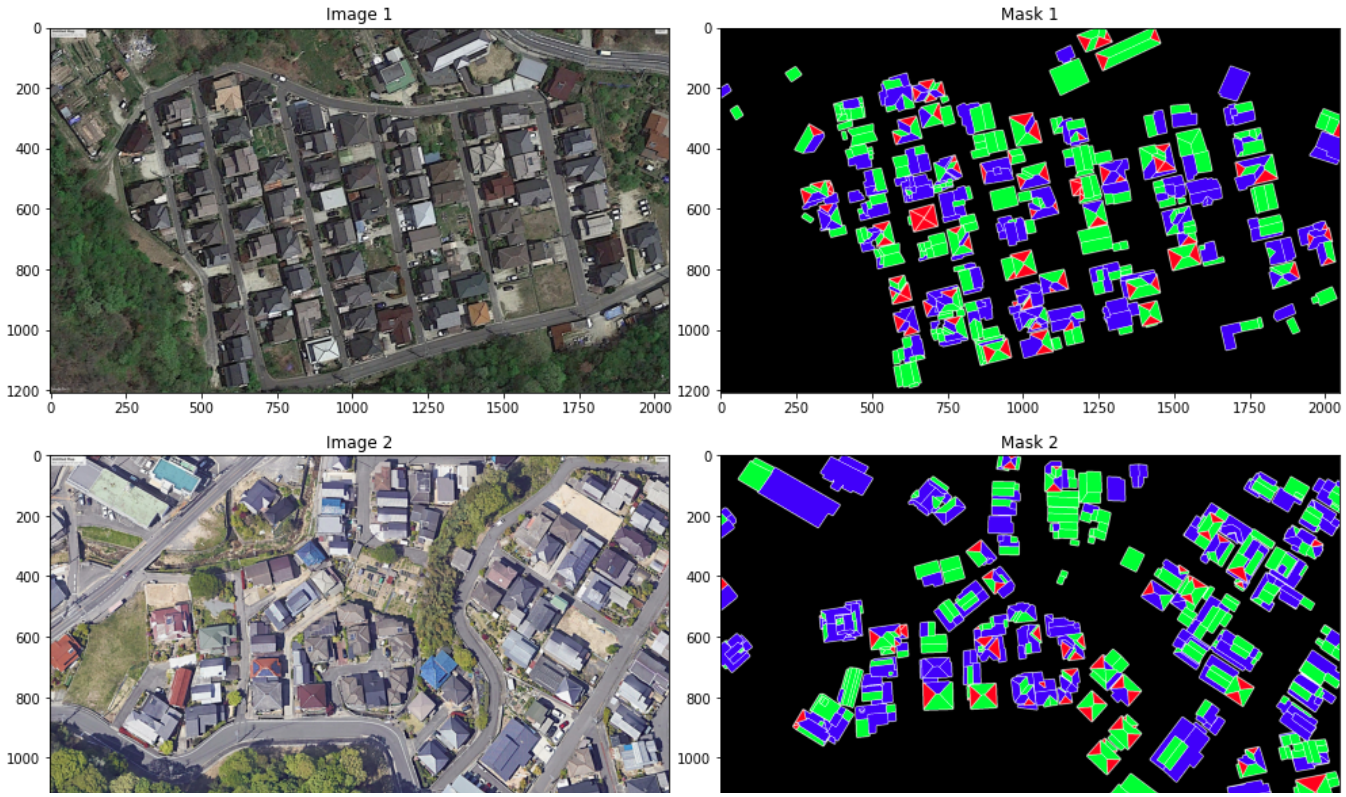
```
inp_img_dir = './input_dir/images'
inp_msk_dir = './input_dir/masks'
out_dir = './output_dir'
height = 512
width = 512
start_num = 1
```

Let's form a list of files with original images and masks and split them:

```
input_images_list = glob.glob(inp_img_dir + '/*.jpg')
input_masks_list = glob.glob(inp_msk_dir + '/*.png')
split(inp_img_dir, inp_msk_dir, out_dir, height, width, start_num)
```

As an example, two original images and masks are shown using the following code:

```
for i, (image_path, mask_path) in enumerate(zip(input_images_list,
                                                input_masks_list)):
    fig, [ax1, ax2] = plt.subplots(1, 2, figsize=(18, 9))
    image = mpimg.imread(image_path)
    mask = mpimg.imread(mask_path)
    ax1.set_title('Image ' + str(i+1))
    ax1.imshow(image)
    ax2.imshow(mask)
    ax2.set_title('Mask ' + str(i+1))
```



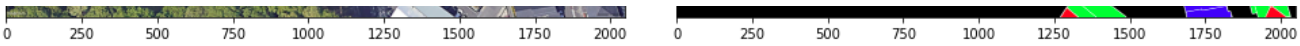


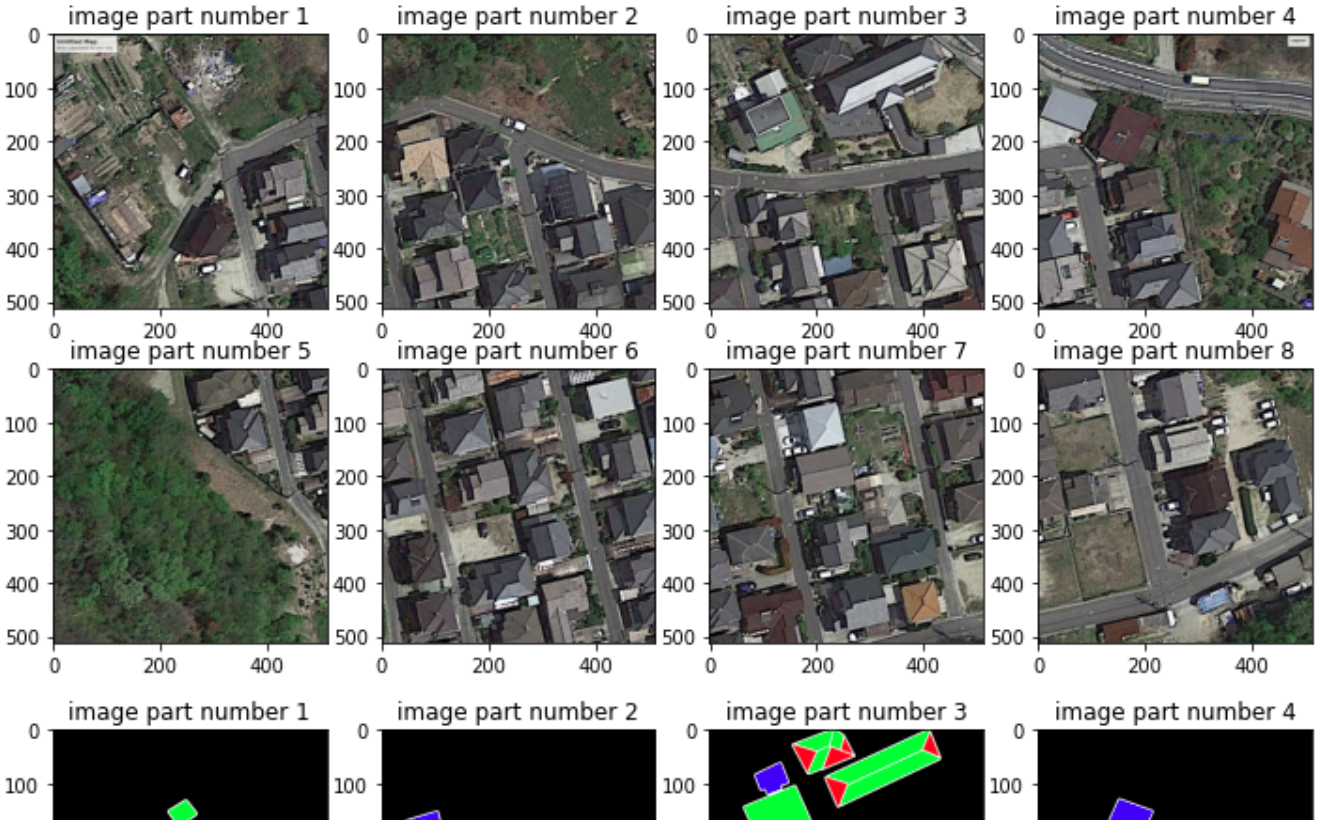
Image is created by Oleksii Sheremet with matplotlib module

Using the following function, you can show all parts of the splitted image (divided into 8 parts):

```
def image_part_plotter(images_list, offset):
    fig = plt.figure(figsize=(12, 6))
    columns = 4
    rows = 2
    # ax enables access to manipulate each of subplots
    ax = []
    for i in range(columns*rows):
        # create subplot and append to ax
        img = mpimg.imread(images_list[i+offset])
        ax.append(fig.add_subplot(rows, columns, i+1))
        ax[-1].set_title("image part number " + str(i+1))
        plt.imshow(img)
    plt.show() # Render the plot
```

Let's see what we got as result of images and masks splitting. For the first image:

```
image_part_plotter(output_images_list, 0)
image_part_plotter(output_masks_list, 0)
```



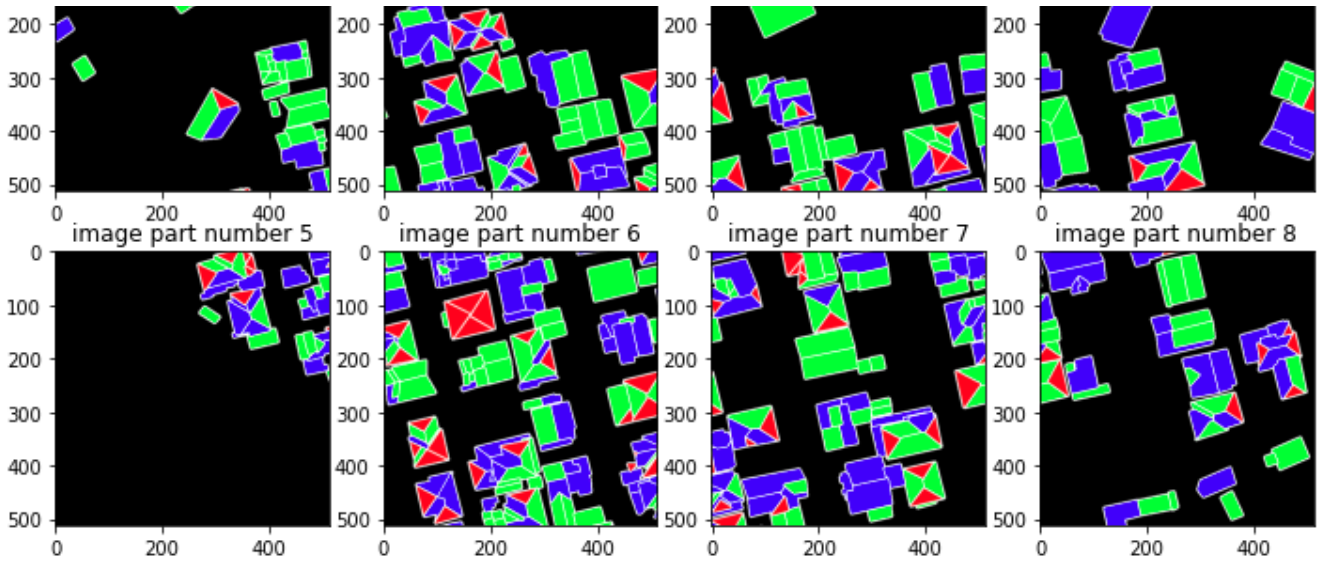
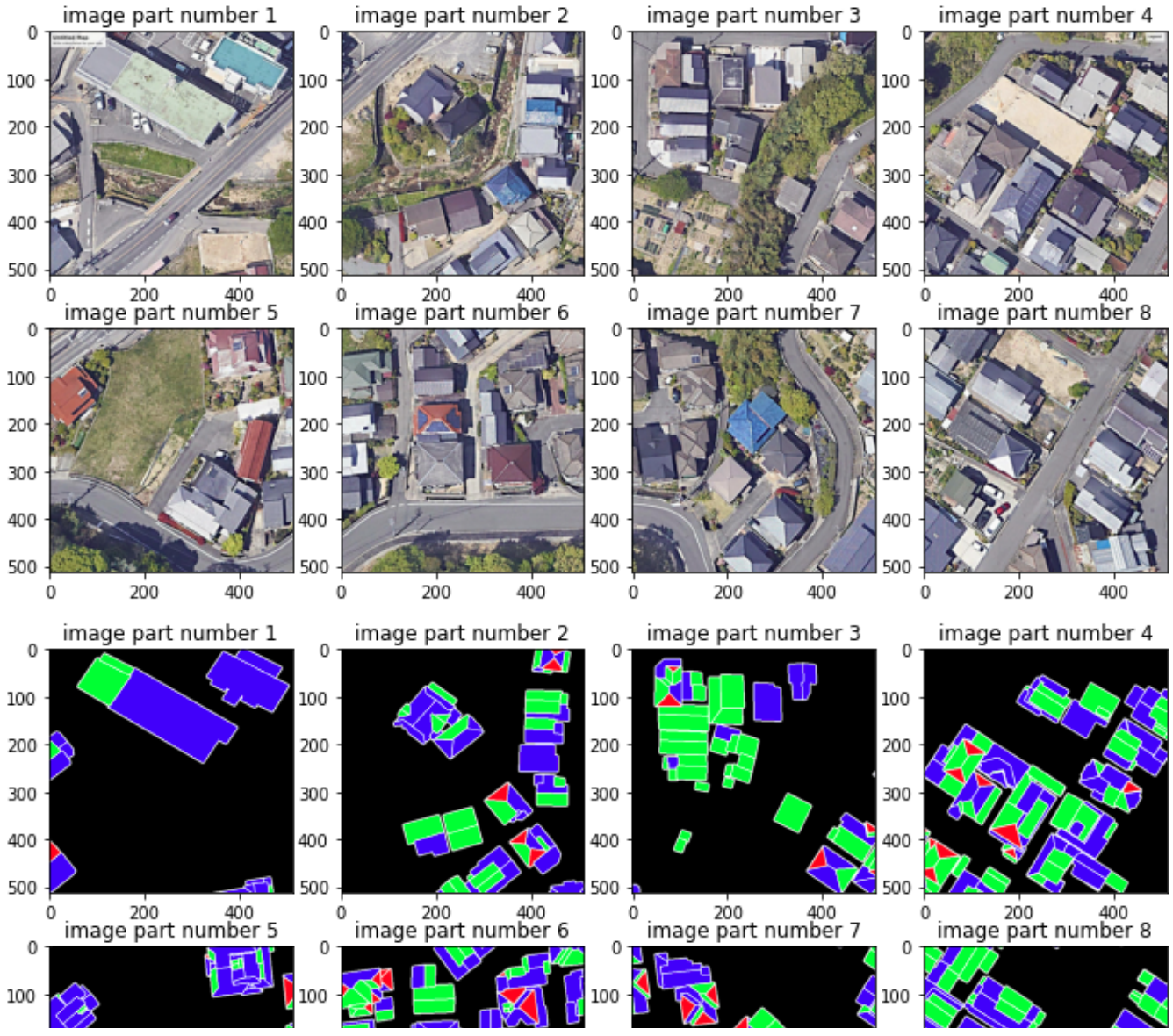


Image is created by Oleksii Sheremet with matplotlib module

```
image_part_plotter(output_images_list, 8)
image_part_plotter(output_masks_list, 8)
```



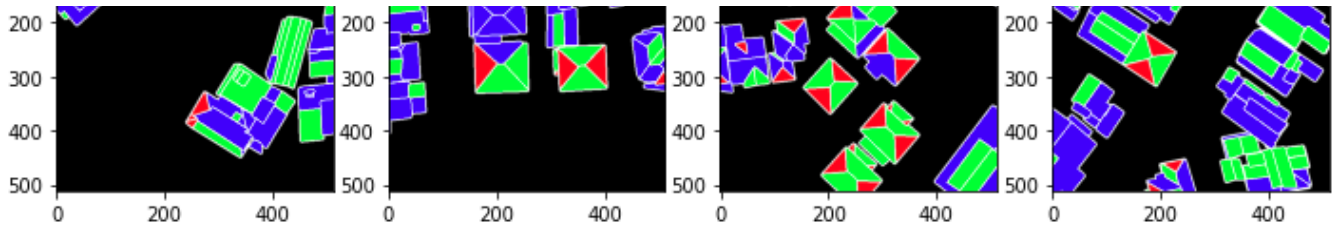


Image is created by Oleksii Sheremet with matplotlib module

Conclusion

The images and masks splitted using the proposed approach are saved with the same file names in different directories, that is, if the file *'./output_dir/images/1_00001.png'* is in the folder with images, then the file *'./output_dir/masks/1_00001.png'* will correspond to it in the directory with masks. After splitting the image and mask, you can apply augmentation to each part of it (for example, change brightness, contrast, rotate or flip). To do this, just add the augmentation function.

References

Pillow (PIL Fork)

Computer Vision Annotation Tool (CVAT)

This notebook provides recipes for loading and saving data from external sources

Annotation Tools Masks Splitting Image Segmentation Computer Vision