Intro to OpenCV CUDA

From single image to Dask Delayed (Python)

Winston Robson Follow
Oct 26, 2020 · 3 min read ★

```python
import cv2 as cv

gpu_frame = cv.cuda_GpuMat()

screenshot = cv.imread('media/drip.png')

gpu_frame.upload(screenshot)

gpu_frame.download()
```

open_cvcuda.py hosted with ❤ by GitHub view raw

Looks like we're stuck in RGB.

Outline

- On a Single Image
- On a Series of Images
- On Series of Images in Parallel with Dask Delayed

On a Single Image

First, we need to create GPU space (`gpu_frame`) to hold an image (as a picture frame holds a picture) before we can upload our image to the GPU.

Step 1: Upload

```
import cv2 as cv

gpu_frame = cv.cuda_GpuMat()
```

opencv_gpu_space.py hosted with ❤ by GitHub view raw

Next, load the image into memory with CPU (`screenshot`), and `.upload()` it to the `gpu_frame` (frame the image);

```
screenshot = cv.imread('media/drop.png')

gpu_frame.upload(screenshot)
```

load_cpu_upload_gpu.py hosted with ❤ by GitHub view raw

Image now in frame, we can start having fun.

Step 2: Have Fun

OpenCV CUDA functions return `cv2.cuda_GpuMat` (GPU matrices), so each result can be operated on without the user having to re- `.upload()` .

Let’s convert the image from RGB to BGR (OpenCV format), then resize it;

```
screenshot = cv.cuda.cvtColor(gpu_frame, cv.COLOR_RGB2BGR)

screenshot = cv.cuda.resize(screenshot, (400, 400))
```

bgr_resize.py hosted with ❤️ by **GitHub** view raw

> *Note: the first function you call should be on the GPU matrix (* `gpu_frame` *) itself, not the image you've just uploaded. This will return a new GPU matrix.*
>
> *The original GPU matrix (* `gpu_frame` *) will continue to hold the original image until a new image is* `.upload()` *ed.*

Step 3: Download

Now you might be wondering, "where's the image?"

Well, it's stuck on the GPU. We need to `.download()` it back to the CPU;

```
screenshot.download()
```

download_cuda_cv.py hosted with ❤️ by **GitHub** view raw

Code to Reproduce this Image

> *Note:* `.download()` *converts the image from* `cv.cuda_GpuMat` *to* `numpy.ndarray`*.*

On a Series of Images

To process a new picture, you can simply `.upload()` that new picture to your existing GPU matrix. Images still must be loaded on CPU before being passed to GPU.

This time we added an inverted binary `.threshold()` to the preprocessing, here's how they came out;

Code to Reproduce this Image

On Series of Images in Parallel with Dask Delayed

With Dask Delayed we can push the above loop into a Dask Delayed function and preprocess multiple series of images in parallel.

I also added in a 2nd `.cvtColor()` to grayscale the images, and switched the inverted binary threshold to a binary threshold.

Defining 2 lists of image files, we can now `.compute()` them side by side;

And here's how they came out;

Code to Reproduce this Image

Fin

You can find the original images used in this story here on GitHub.

Thanks for reading. Please feel free to respond with any questions.

Dropout-Analytics/opencv_cuda

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or or…

github.com

Continued Reading

OpenCV CUDA for Videos

No camera required.

medium.com

Set up and get started (+ test code)

medium.com

Beginner’s Guide to KNN with cuML

What is K-Nearest Neighbors? And how to implement it in Python with RAPIDS cuML

medium.com

References

Koriukina, Valeriia. “Getting Started with OpenCV CUDA Module.” *Learn OpenCV*, Learnopencv.com, 15 Sept. 2020, learnopencv.com/getting-started-opencv-cuda-modul.

McWhorter, Paul. “AI on the Jetson Nano LESSON 10: Installing OpenCV for Python 3.” *Paul McWhorter — YouTube*, Youtube.com/User/Mcwhorpj, 2 Nov. 2019, youtu.be/3QYayL5y2hk.

Pulli, Kari; Baksheev, Anatoly; Kornyakov, Kirill; Eruhimov, Victor. “Realtime Computer Vision with OpenCV.” *Realtime Computer Vision with OpenCV — ACM Queue*, Association for Computing Machinery, 22 Apr. 2012, queue.acm.org/detail.cfm?id=2206309.

Opencv Cuda Image Processing Data Science Computer Vision