Problem: Use gpu hardware accelerators with opencv for decoding You may have used opencv to load and run inference on a video. However opencv, by default will not use GPU to decode the video. Your nvidia GPU has dedicated hardware decoders and encoders for efficiently encoding or decoding the videos.

My solution to this problem is of using UMat for the task of loading of frames from the video

But what is UMat exactly??

The UMat class tells OpenCV functions to process images with an OpenCL specific code which uses an OpenCL-enabled GPU if it exists in the system (automatically switching to CPU otherwise). The UMat internally implements OpenCL framework which tries to harness the processing power of any capable hardware attached to the device, it could be CPU, GPU or even Digital Signal Processor present in mobile devices, so OpenCL performs efficient multi-processing along various available devices capable of processing.

So now lets see some codes and screen shots

```python
import cv2
import timeit

# A simple image pipeline that runs on both Mat and Umat
def img_cal(img, mode):
    if mode=='UMat':
        img = cv2.UMat(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.GaussianBlur(img, (7, 7), 1.5)
    img = cv2.Canny(img, 0, 50)
    if type(img) == 'cv2.UMat':
        img = cv2.UMat.get(img)
    return img

# Timing function
```

```python
def run(processor, function, n_threads, N):
    cv2.setNumThreads(n_threads)
    t = timeit.timeit(function, globals=globals(), number=N)/N*1000
    print('%s avg. with %d threads: %0.2f ms' % (processor, n, t))
    return t

img = cv2.imread('file_example_TIFF_10MB.tiff')
img_UMat = cv2.UMat(img)
N = 1000
n_threads = [1,  16]

processor = {'GPU': "img_cal(img_UMat,'UMat')",
             'CPU': "img_cal(img,'Mat')"}
results = {}
for n in n_threads:
    for pro in processor.keys():
        results[pro,n] = run(processor=pro,
                             function= processor[pro],
                             n_threads=n, N=N)

print('\nGPU speed increase over 1 CPU thread [%%]: %0.2f' % \
      (results[('CPU', 1)]/results[('GPU', 1)]*100))
print('CPU speed increase on 16 threads versus 1 thread [%%]: %0.2f' % \
      (results[('CPU', 1)]/results[('CPU', 16)]*100))
print('GPU speed increase versus 16 threads versus 1 thread [%%]: %0.2f' %
\
      (results[('GPU', 1)]/results[('GPU', 16)]*100))
```

In this chunk i have done nothing fancy just changes the ordinary mat form to UMat form (in the underlined part)

Now some results of this code

A comparison was made between the GPU and CPU processing time of a 10MB OpenCV pipeline that consisted of a conversion to grayscale, Gaussian blur, then a Canny edge detector. The pipeline was run 1000 x then averaged on the GPU then compared to a single thread and to all 16 threads on the CPU running at 3.60 GHz.

The results:

GPU avg. with 1 threads: 6.98 ms

CPU avg. with 1 threads: 57.69 ms

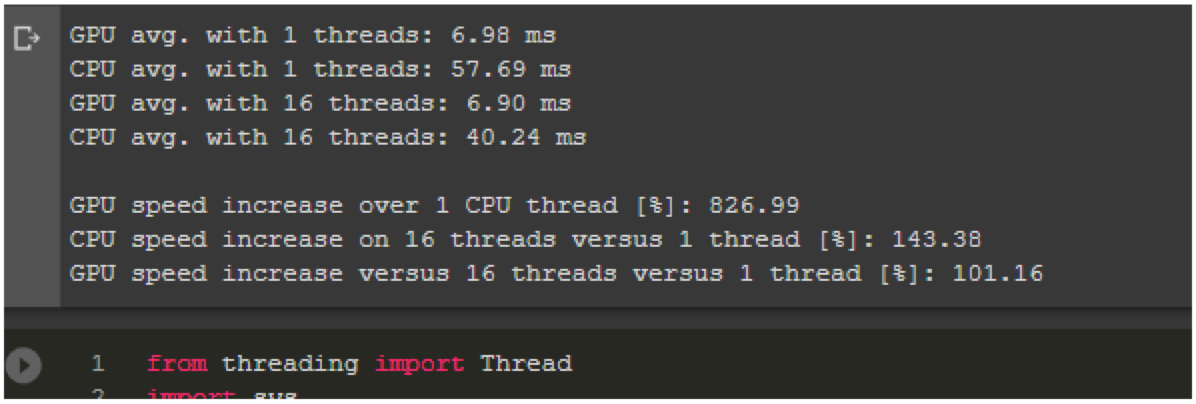
GPU avg. with 16 threads: 6.90 ms

CPU avg. with 16 threads: 40.24 ms

GPU speed increase over 1 CPU thread [%]: 826.99

CPU speed increase on 16 threads versus 1 thread [%]: 143.38

GPU speed increase versus 16 threads versus 1 thread [%]: 101.16



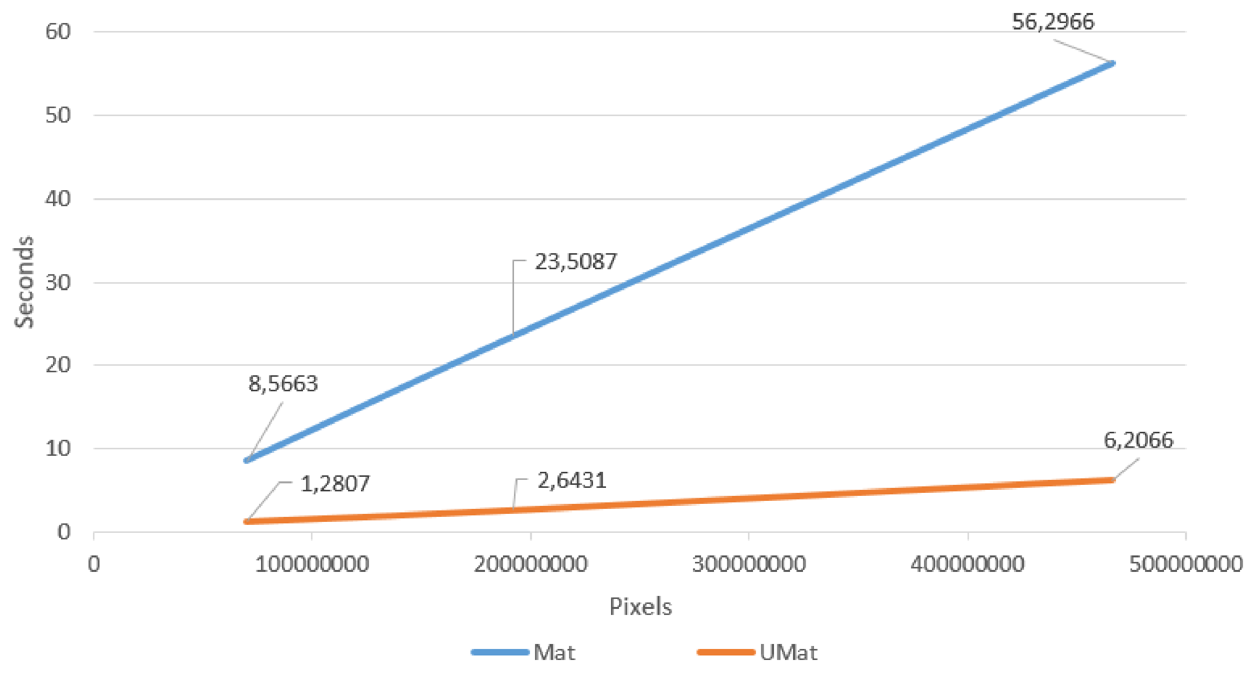
```
GPU avg. with 1 threads: 6.98 ms
CPU avg. with 1 threads: 57.69 ms
GPU avg. with 16 threads: 6.90 ms
CPU avg. with 16 threads: 40.24 ms

GPU speed increase over 1 CPU thread [%]: 826.99
CPU speed increase on 16 threads versus 1 thread [%]: 143.38
GPU speed increase versus 16 threads versus 1 thread [%]: 101.16
```

```python
from threading import Thread
```

So here we can clearly see how UMat can speed the task

Also found this amazing graph from google that shows how the performance difference is between UMat and Mat



Now let's come to the part of how to use this on video processing

```
import cv2
from queue import Queue

class UMatFileVideoStream:
    def __init__(self, path, queueSize=128):
        self.stream = cv2.VideoCapture(path)
        self.stopped = False
        self.count = 0
        self.Q = Queue(maxsize=queueSize)
        self.width = int(self.stream.get(cv2.CAP_PROP_FRAME_WIDTH))
        self.height = int(self.stream.get(cv2.CAP_PROP_FRAME_HEIGHT))
        self.frames = [0] * queueSize
        for i in range(queueSize):
            self.frames[i] = cv2.UMat(self.height, self.width,
cv2.CV_8UC3)
    def __del__(self):
        self.stream.release()
    def start(self):
        t = Thread(target=self.update, args=())
        t.daemon = True
        t.start()
        return self
    def update(self):
        while True:
            if self.stopped:
                return
            if not self.Q.full():
                self.count += 1
                target = (self.count-1) % self.Q.maxsize
                grabbed = self.stream.grab()
                if not grabbed:
                    self.stop()
                    return
                self.stream.retrieve(self.frames[target])
                self.Q.put(target)

    def read(self):
        while (not self.more() and self.stopped):
            sleep(0.1)
        return self.frames[self.Q.get()]
    def more(self):
        return self.Q.qsize() > 0
    def stop(self):
        self.stopped = True
```

This code is adapted from link

```
%%timeit
import cv2
video = UMatFileVideoStream('file_example_MP4_640_3MG.mp4').start()

while not video.stopped:
    img = cv2.cvtColor(video.read(), cv2.COLOR_BGR2RGB, 0)
    # more of processing before fetching the images
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.GaussianBlur(img, (7, 7), 1.5)
    img = cv2.Canny(img, 0, 50)
    img = cv2.UMat.get(img)    # image is now a numpy array
```

1 loop, best of 3: 1min 21s per loop

So it takes 1min 21s to read a file and do these operation on a 17MB and 640 resolution file with UMat

While it just takes 2 minutes per loop for reading same file on without UMat

Websites that get this done

1. https://stackoverflow.com/questions/38809954/does-opencv-umat-always-reside-on-gpu
2. https://www.pyimagesearch.com/2016/07/11/compiling-opencv-with-cuda-support
3. https://www.youtube.com/watch?v=TT3_dlPL4vo
4. https://www.youtube.com/watch?v=-QVTXdHEzzs

In the end i want to thank you for the task i learn a lot from it and will wait for next round

Thank you

Pranshu Rastogi