**1. 테스트 개요**   
OpenCV와 직접 구현한 CustomDLL의 Blur 기능을 다음 세 가지 측면에서 비교: 처리 속도, 메모리 사용량, 결과 영상의 일치 여부

**2. 테스트 환경**

**하드웨어 사양:**

* CPU: AMD Ryzen 9 7950X3d 16-Core Processor
* GPU: NVIDIA GTX 4090
* RAM: 32GB

**소프트웨어 환경:**

* OS: Windows 10 Pro 64-bit
* Development Tool: Microsoft Visual Studio 2022
* OpenCV Version: 4.9.0

**3. 테스트 방법**

**입력 이미지 데이터:**

* 채널 수: 1 (Gray Scale)
* 이미지 수: 3장
* 크기 : (8763 x 11648), (8256 x 5504), (6759 x 4508)

**테스트 절차:**

1. 입력 이미지를 Blur 처리
2. 처리 시간을 측정 (Processing Time)
3. 최대 및 평균 메모리 사용량을 측정 (Memory Usage)
4. 결과 영상의 SSIM(Structural Similarity Index Map) 및 MSE(평균 제곱 오차) 확인

**4. 테스트 결과**

|  |  |  |  |
| --- | --- | --- | --- |
| **항목** | **Kernel Size** | **OpenCV** | **Custom** |
| **실행 시간 (ms)** | 30 | 1341 ms | 3581 ms |
| 31 | 1304 ms | 3514 ms |
| **평균 메모리 사용량 (MB)** | 30 | 179.133 MB | 977.968 MB |
| 31 | 178.852 MB | 994.312 MB |
| **최대 메모리 사용량 (MB)** | 30 | 362.512 MB | 2065.54 MB |
| 31 | 362.566 MB | 2065.54 MB |
| **SSIM** | 30 | 0.999956, 0.999955, 0.999968 | |
| 31 | 전부 이미지가 일치함 | |
| **MSE** | 30 | 0.00315725, 0.00343216, 0.00213039 | |
| 31 | 전부 이미지가 일치함 | |

**5. 분석 및 결론**

1. **실행 시간:** Custom의 처리 속도가 OpenCV보다 약 1.6배 느리게 나타났습니다.
2. **Memory Usage:** 평균 메모리 사용량은 Custom이 OpenCV 보다 약 3.6배 높았습니다. 최대 메모리 사용량은 Custom이 OpenCV 보다 약 5.7배 높았습니다.
3. **결과 영상 일치:** Kernel Size가 홀수일 경우에는 완전히 일치했지만, 짝수일 경우에는 완전히 일치하지는 않았습니다. 하지만 육안으로 확인할 수는 없었고, 픽셀 값 간의 차이도 1, 2 정도밖에 나지 않았습니다. SSIM, MSE역시 두 이미지는 거의 동일하다는 결과를 내놓았습니다.
4. **최종 결론 :** OpenCV은 속도 및 메모리 효율성에서 훨씬 더 우수합니다. 메모리 사용량의 경우는, 제가 구현한 것이 메모리를 더 많이 사용한 것은 어느정도 예상 했습니다. 왜냐하면 저의 경우는 원본 이미지 데이터, 출력 이미지 데이터 말고도 커널 크기의 절반만큼 반사된 이미지의 데이터와 그 데이터의 누적 합을 가지고 있는 데이터 4개를 사용했기 때문입니다. 반사된 이미지의 데이터 없이도 커널에서의 평균을 O(1)시간만에 구할 수 있게 만들 아이디어는 있었지만, 이렇게 하면 구현도 조금 더 복잡해지고, 같은 상수시간이지만 시간도 약간 늘어날 것이라 생각해서 메모리는 포기하고 시간을 더 집중적으로 생각했습니다.   
   그런데 시간의 경우도 OpenCV의 Blur보다 느렸습니다. 제 생각에는 OpenCV는 소수점의 정밀도를 낮추거나 SIMD 최적화 기법을 사용해 성능을 높인 것으로 생각됩니다. 그리고 코드에서 반올림, clamp를 std라이브러리에 있는 것을 사용하였을 때 보다 cv::saturate\_cast<uchar>를 사용하는 것이 사용 시간이 2/3으로 줄어들고, 평균 메모리 사용량은 OpenCV와 비슷해진 것으로 보아, cv::saturate\_cast<uchar>에도 최적화가 되어 있다는 것을 짐작할 수 있습니다. 그리고 커널의 크기가 짝수일 경우에는 이미지가 거의 비슷했지만 완전히 일치하지는 않았는데, cv::saturate\_cast<uchar>를 사용해 보니 이미지가 일치하였습니다. 하지만 외부 라이브러리는 사용하지 않는다는 조건이기 때문에 결과물 코드에는 cv::saturate\_cast<uchar>이 사용되지 않았습니다.