WebGPU Image Super Resolution

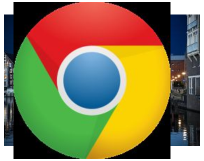
Andrew Zhu
andrewyx@seas.upenn.edu
https://github.com/Andrewzhuyx

Paul (San) Jewell
pjewell@pennmedicine.upenn.edu
https://github.com/sona1111

https://bit.ly/3ysr8xC

Project Overview

- A WebGPU based image super resolution program
- Input image is fed to a neural network to generate the output



DEMO

https://bit.ly/3ysr8xC

https://bit.ly/3ysr8xC

Super Resolution Neural Network Architecture

RGB Channels

X

1	2	3
-4	7	4
2	-5	1

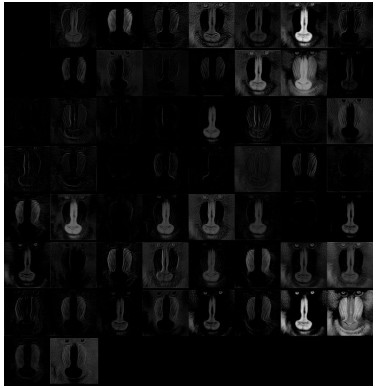
Filter / Kernel

=

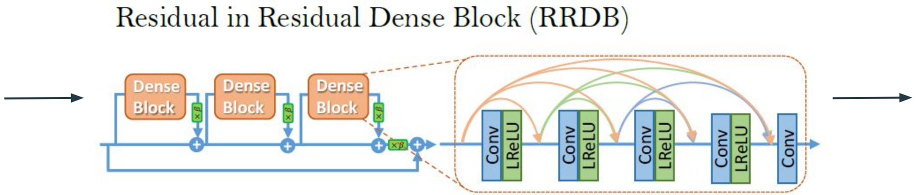
64 Channels

https://bit.ly/3ysr8xC

Super Resolution Neural Network Architecture



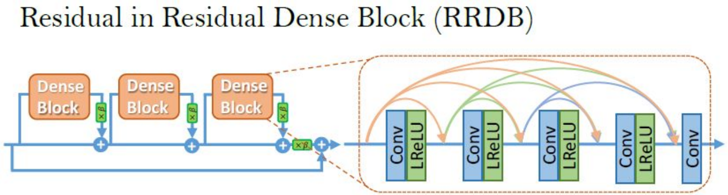
64 Channels



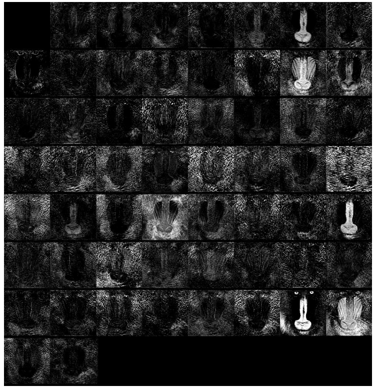
64 Channels

Super Resolution Neural Network Architecture

64 Channels



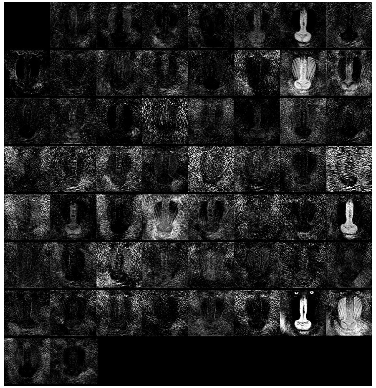
X 23



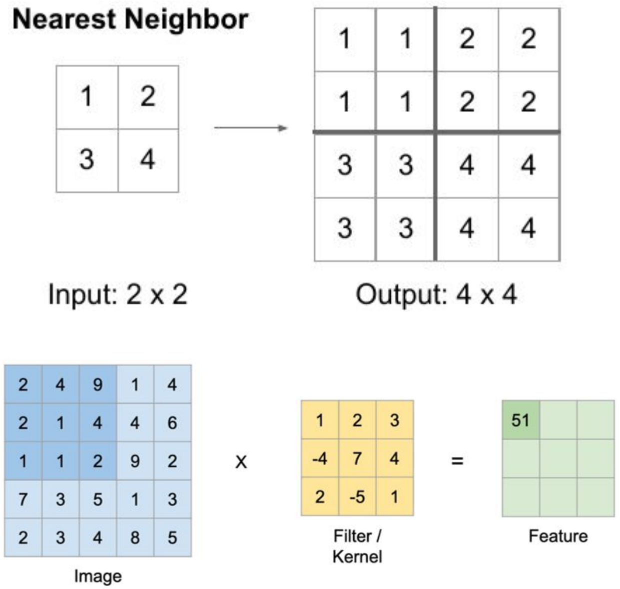
64 Channels

https://bit.ly/3ysr8xC

Super Resolution Neural Network Architecture



64 Channels

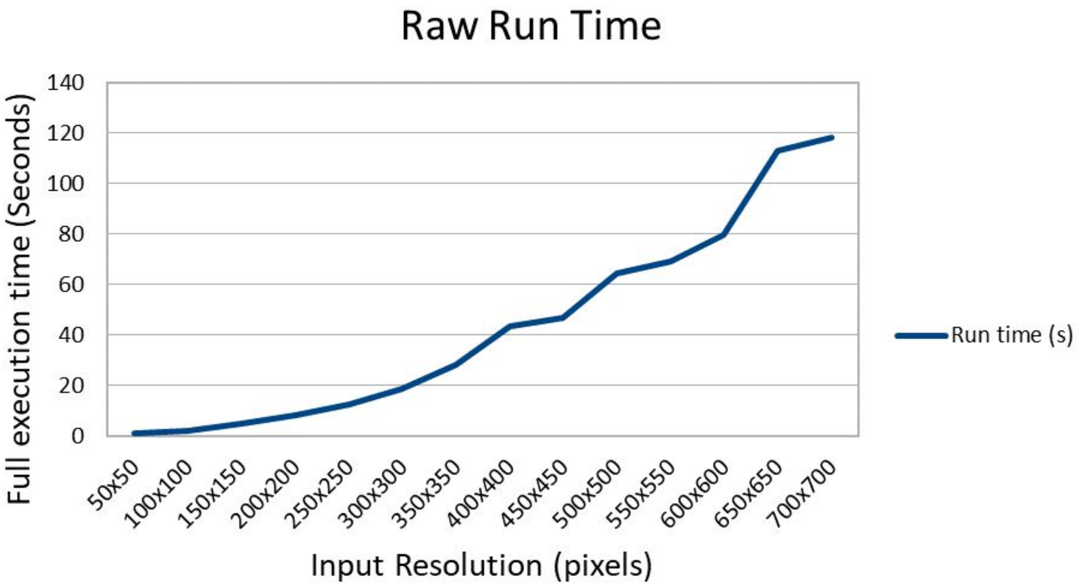


X 2

RGB Channels

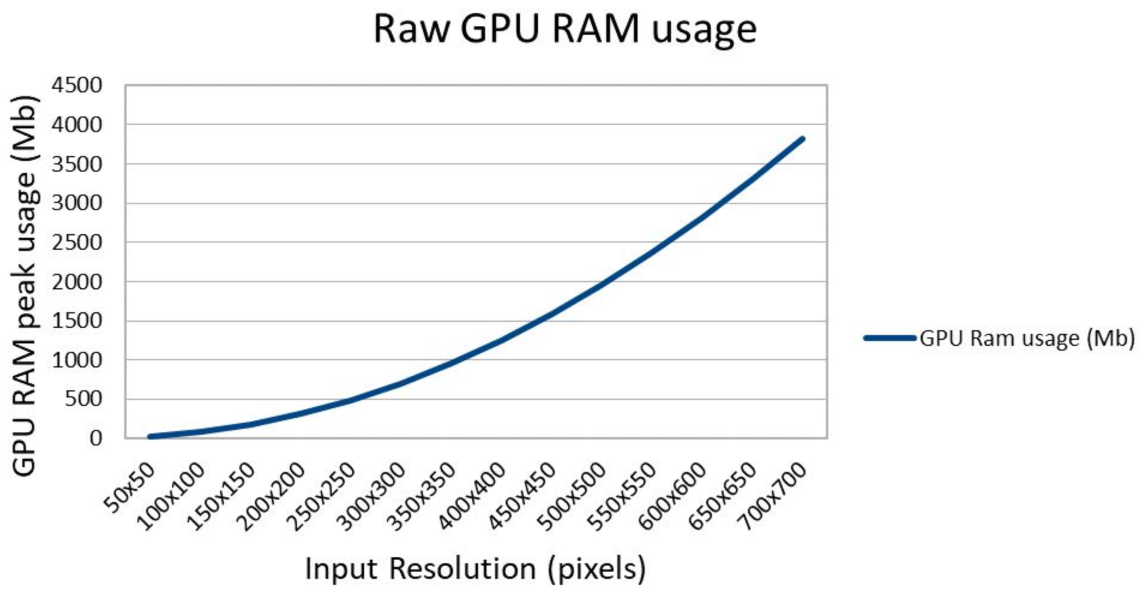
https://bit.ly/3ysr8xC

Performance Analysis



https://bit.ly/3ysr8xC

Performance Analysis



https://bit.ly/3ysr8xC

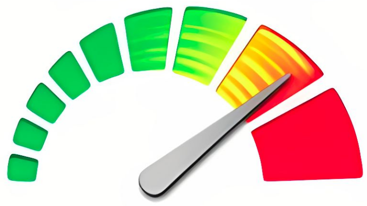
Performance Improvements

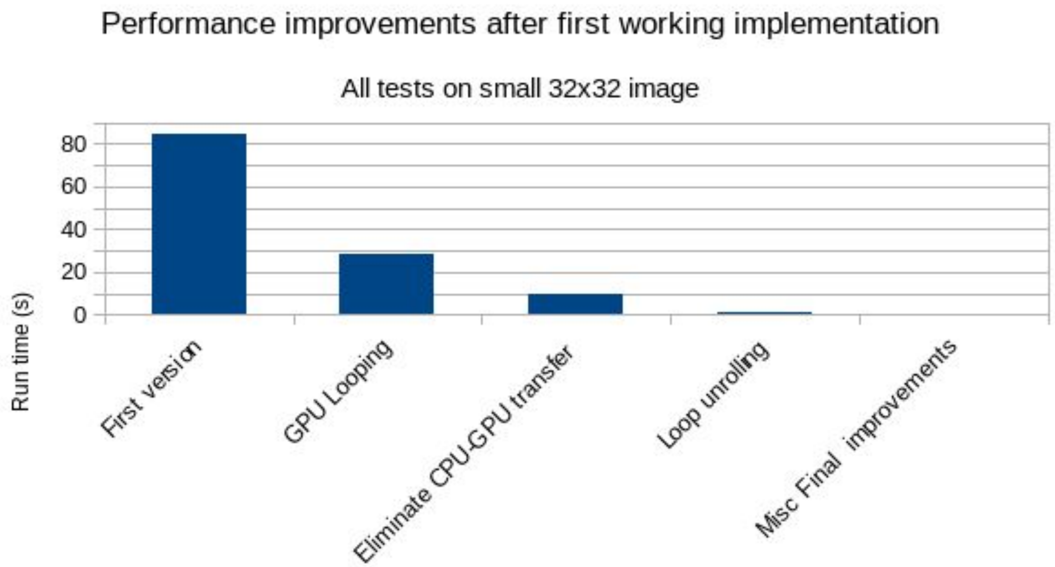
From a few minutes to 5 seconds!

- Removed all data transfer from CPU <> GPU between kernel calls
- Minimized the number of memory buffers
- Manual loop unrolling inside kernels
- Loaded weight and bias data to GPU as it is downloaded to browser.
- Used IndexDB web API to cache neural network weights and biases.

https://bit.ly/3ysr8xC

Performance Improvements





https://bit.ly/3ysr8xC

Challenges

- Actively developed standard -- performance and bug occurrence varies from day to day throughout this project due to patches in chrome and firefox!
- Matching PyTorch algorithms exactly was tedious and non-trivial.
  - We overcame this challenge partially by re-implementing the model in python before switching to WebGPU
- Relatively low resolution limit for hardware -- 700x700 image requires ~4GB to process!
  - Still not yet resolved

https://bit.ly/3ysr8xC

Potential further Improvements

- Allow larger input images and maximize performance
  - Int8 quantization (unsure if performance will be acceptable for this kind of network)
  - Float16 quantization (not natively supported with webGPU), workarounds?
  - Tiled implementation with overlap
- Multi-network simultaneous evaluation
- More pre-trained networks
- Browser extension / plugin

https://bit.ly/3ysr8xC

Acknowledgements

- Shrek Shao, chrome webGPU team
- Austin Eng, chrome webGPU team
- Dzmitry Malyshau, firefox webGPU team
- Shehzan Mohammed, course professor
- Liam Dugan, course teaching assistant
- Janine Liu, course teaching assistant

WebGPU Image Super Resolution

Andrew Zhu
andrewyx@seas.upenn.edu
https://github.com/Andrewzhuyx

Paul (San) Jewell
pjewell@pennmedicine.upenn.edu
https://github.com/sona1111

https://bit.ly/3ysr8xC

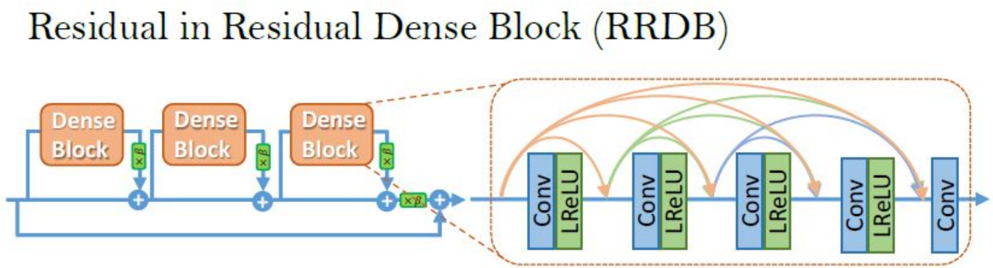
References

- https://web.dev/gpu-compute/
- https://github.com/austinEng/webgpu-samples
- https://github.com/xinntao/ESRGAN
- https://github.com/xinntao/Real-ESRGAN
- https://sahnimanas.github.io/post/anatomy-of-a-high-performance-convolution/
- Wang, Xintao, et al. "Esrgan: Enhanced super-resolution generative adversarial networks." Proceedings of the European conference on computer vision (ECCV) workshops. 2018.
- Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).

https://bit.ly/3ysr8xC

Super Resolution Neural Network Architecture

1. Input: RGB image (3 * height * width)
2. First Convolution Layer (Channel 3 -> 64)
3. 23 x Residual in Residual Dense Block



4. Second Convolution + Residual Layer
5. 2 x Super Resolution Layer (width x 2, height x 2)
6. 2 x Super Resolution Layer (width x 2, height x 2)
7. Last Convolution Layers (Channel 64 -> 3)
8. Output: RGB image (3 * (4 * height) * (4 * width))

https://bit.ly/3ysr8xC

Memory Optimization

```
'''
Convolve `input` with `kernel` to generate `output`
    input.shape = [input_channels, input_height, input_width]
    kernel.shape = [num_filters, input_channels, kernel_height, kernel_width]
    output.shape = [num_filters, output_height, output_width]
'''
for filter in 0..num_filters
    for channel in 0..input_channels
        for out_h in 0..output_height
            for out_w in 0..output_width
                for k_h in 0..kernel_height
                    for k_w in 0..kernel_width
                        output[filter, out_h, out_w] +=
                            kernel[filter, channel, k_h, k_w] *
                            input[channel, out_h + k_h, out_w + k_w]
```

GPU Parallelize

Loop Unrolling