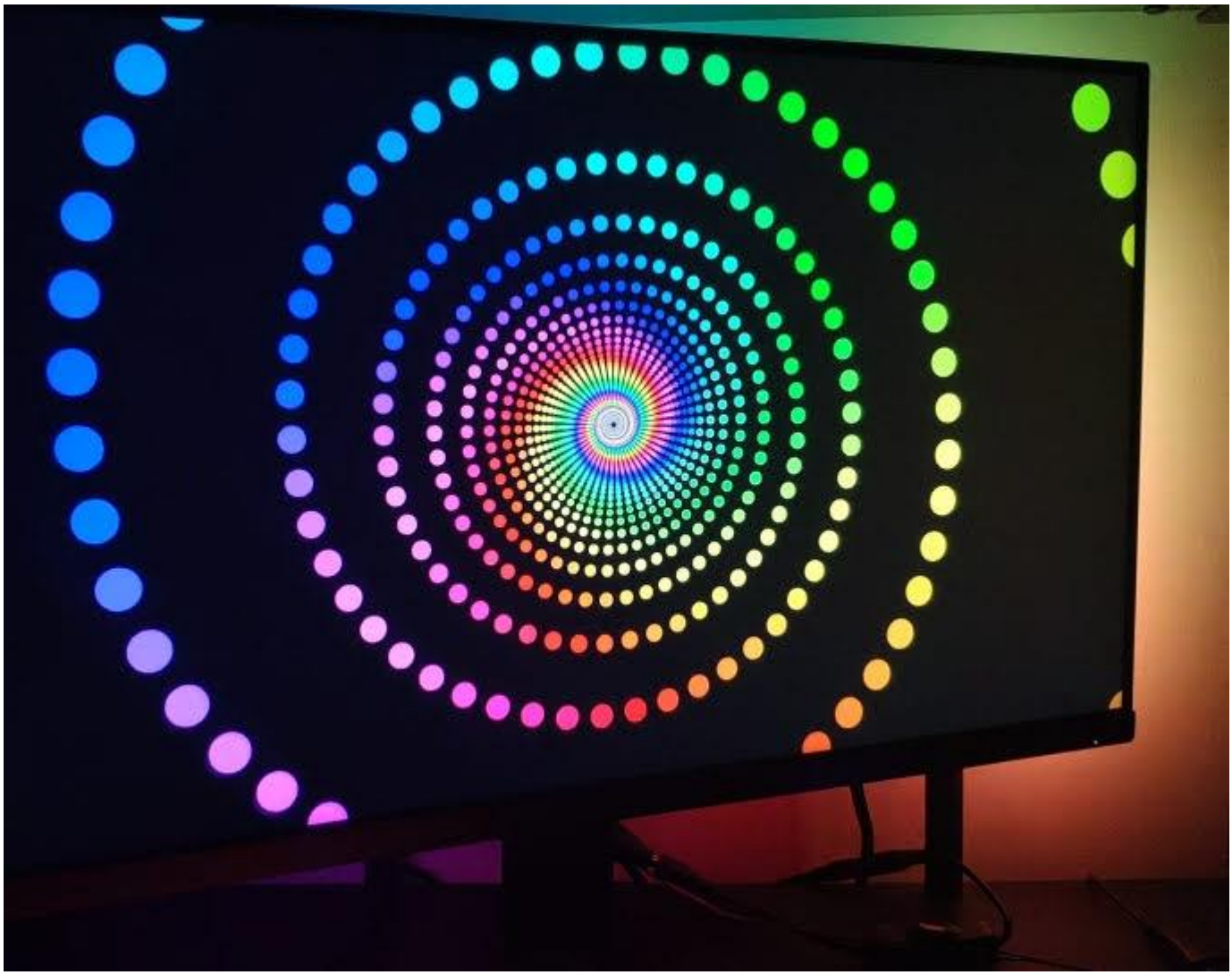
AURALIGHT



19.04.2020
SM2716

SIU KING WAI (54412743)
DR. CHRISTIAN SANDOR

INTRODUCTION

Auralight is a cost-effective solution to extend your entertainment to a next level. It extends the edges of your screen to the surroundings by casting ambient lights on the wall, thus bringing you into the moment. Built on ESP32, it provides a web interface for an easier control. The asynchronous design in coding provides low latency response. It can be used when you're watching movies, playing video games or even reading books.

Auralight provides two modes, the first one is 'extension' mode. It samples the edges of your screen and create an interpolated 'ambient' version; the second one is the 'breathing' light mode, accompanying your reading journey or lighting up the room even the computer is not on. The main control web interface allows user to adjust the brightness scaling when in 'extension' mode and to adjust the color correction tone according to the FastLED API. Apart from that, user can also adjust the breathing/blinking frequency, the start and end of breathing/blinking brightness and the color.

This project covers various technologies, including hardware C++ coding using Arduino IDE, jQuery and Python. I was also glad to study some mathematics during my research of computer vision.

HARDWARE

1. ESP-32
   (~30 HKD)
2. WS2812BECO 2 x 108 LEDs
   (~130 HKD)
3. 5V 8A DC Power Adaptor
   (~40 HKD)

WHY

The idea came to me when I saw an article on the Internet showcasing something called 'Hue', which was an ambient light and it sort of extended the screen. I really wanted something like that, but it was expensive. I knew I would to build it myself. It was even before I start this course, that I already knew what I wanted to do for this course's project.

Philips HUE

Certainly, this was not a groundbreaking idea, and similar works have already been done. Apart from Phillips Hue, there's also a slightly cheaper solution called lightpack. However, their software didn't work well when there were lots of LEDs. I think it would be a nice challenge to build it.

I myself have been using it everyday since I successfully built it, and the performance is better than their software because I used a machine learning framework called Pytorch to do the calculations using GPU. And I was really surprised when it really worked, as I thought it such a heavy framework would only be suitable for doing more intensive tasks.

How

In this project, the strip was cut to 174 LEDs, where the top row has 55 LEDs, bottom row has 59 LEDs, and both sides have 30 LEDs.

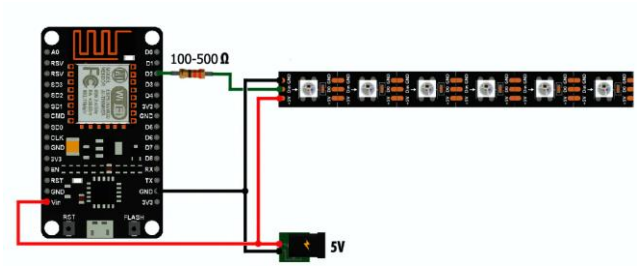
Python libraries:

1. D3DShot
2. Numpy
3. Pytorch
   (CUDA 10.0)

ESP-32 libraries:

1. FastLED
2. AsyncUDP
3. ESPAsyncWebServer

Installation on a 25" monitor



Schematic connecting a WS2812B to an ESP8266. In my project I am using an ESP32 and the LED strip data pin (Green) is connected to PIN 15, resistor is omitted.

Flowchart

Video capture

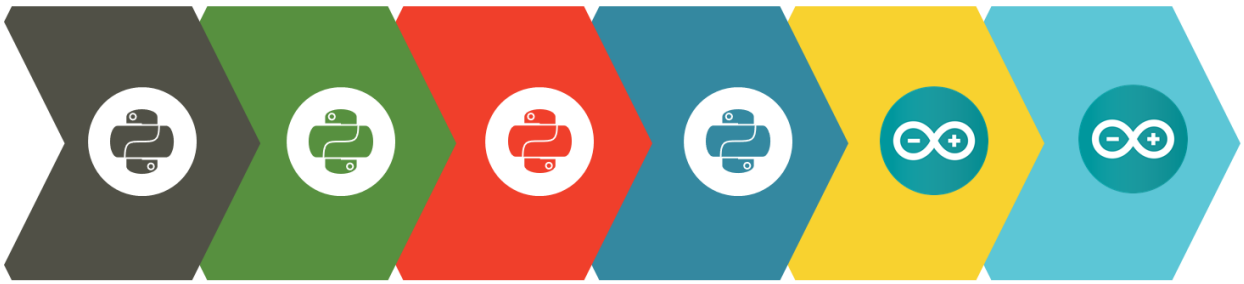
Capturing in a Python program at around 50 fps realistically, and the out is a numpy float(normalized to 0,1) matrix

Matrix flattening

The matrix is flattened to a one dimensional array representing the channel of the LEDs in a chronological order of [R, G, B, R, G, B] as float.

Receive UDP packets

The ESP32 AsyncUDP Server listen to a port, if a packet is found, it parses the packet and then apply the colors to the FastLED Object



Tensor interpolation

The matrix is transformed to a tensor to fit a Pytorch interpolation function, the matrix go through down sampling first, than up sampling to smooth out (blur) the pixels further. Returns back a numpy matrix.

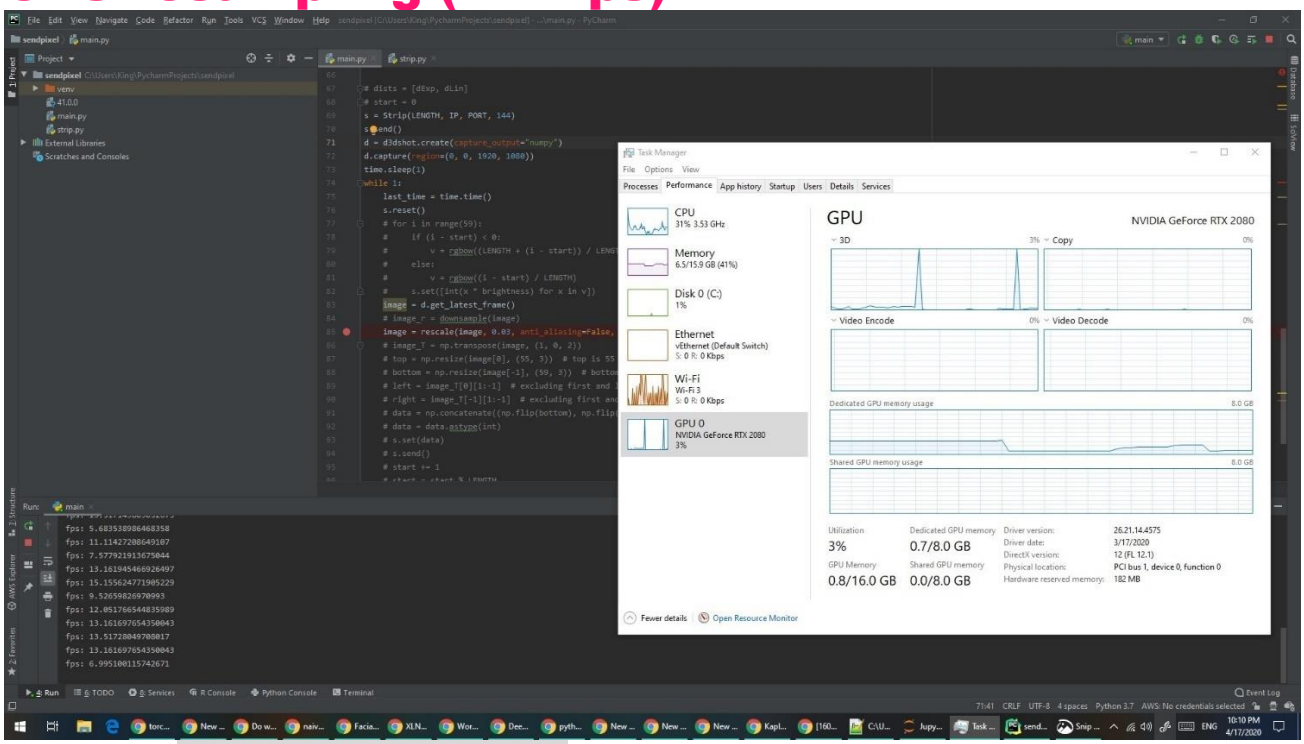
Sending UDP packets

The python program further apply multiplication and offset to convert it back to byte type, then simply send the array by converting it into a byte stream.

Shine bright

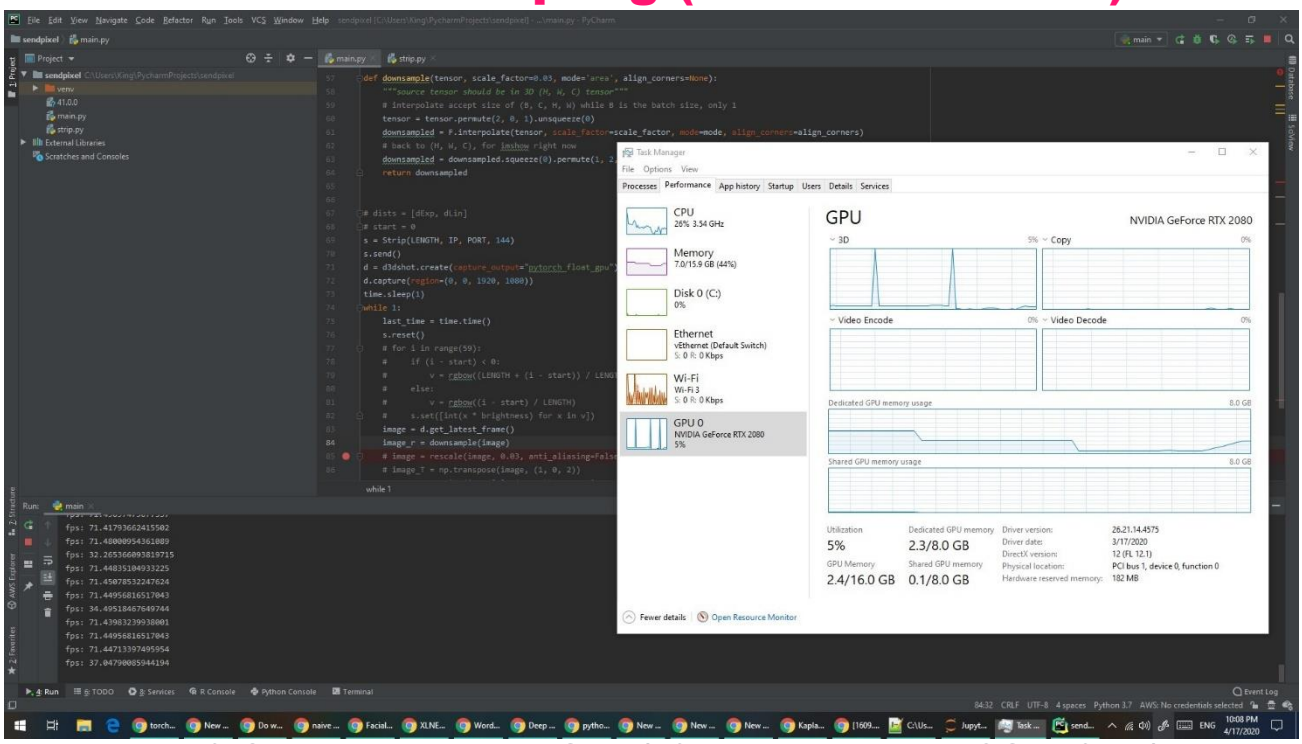
Appreciate the extension screen.

CPU resampling (~12 fps)



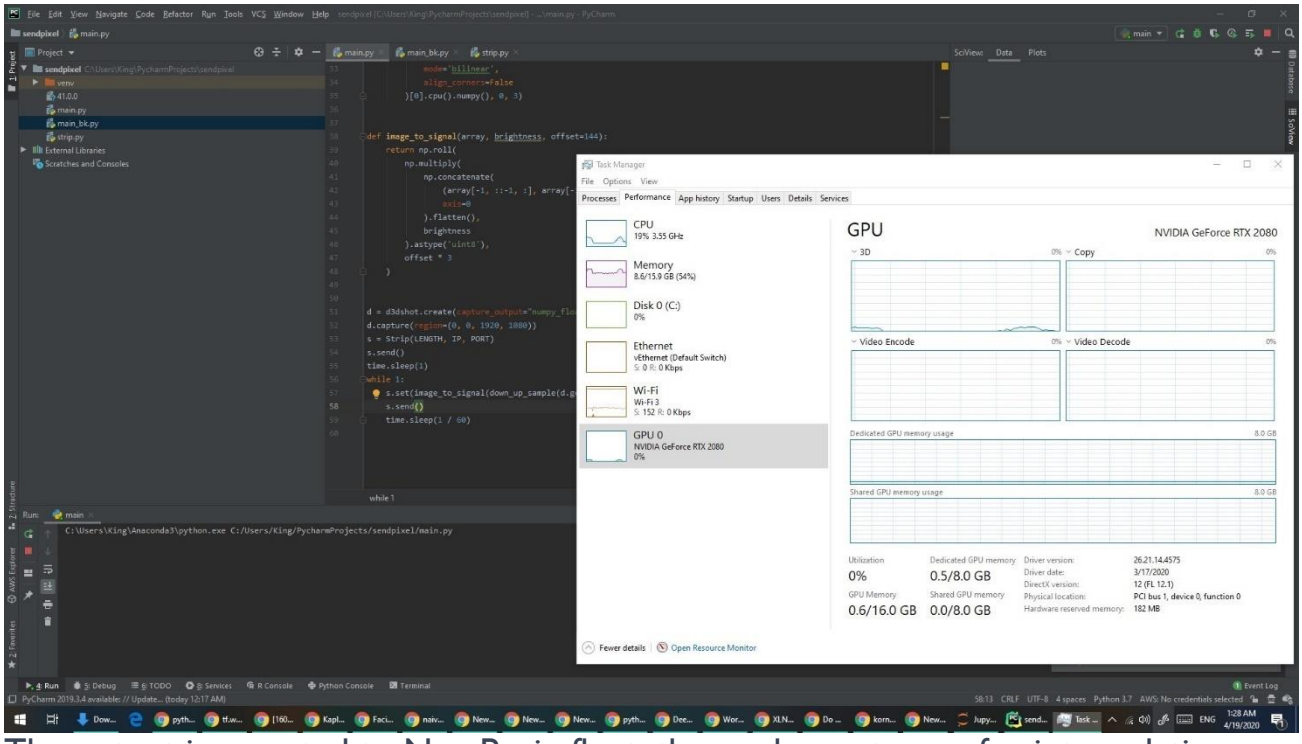
I was using skimage.transform.rescale to perform the resampling.

CUDA Tensor resampling (almost realtime)



The output of d3dshot was Pytorch CUDA float tensor, it took 2GB of VRAM.

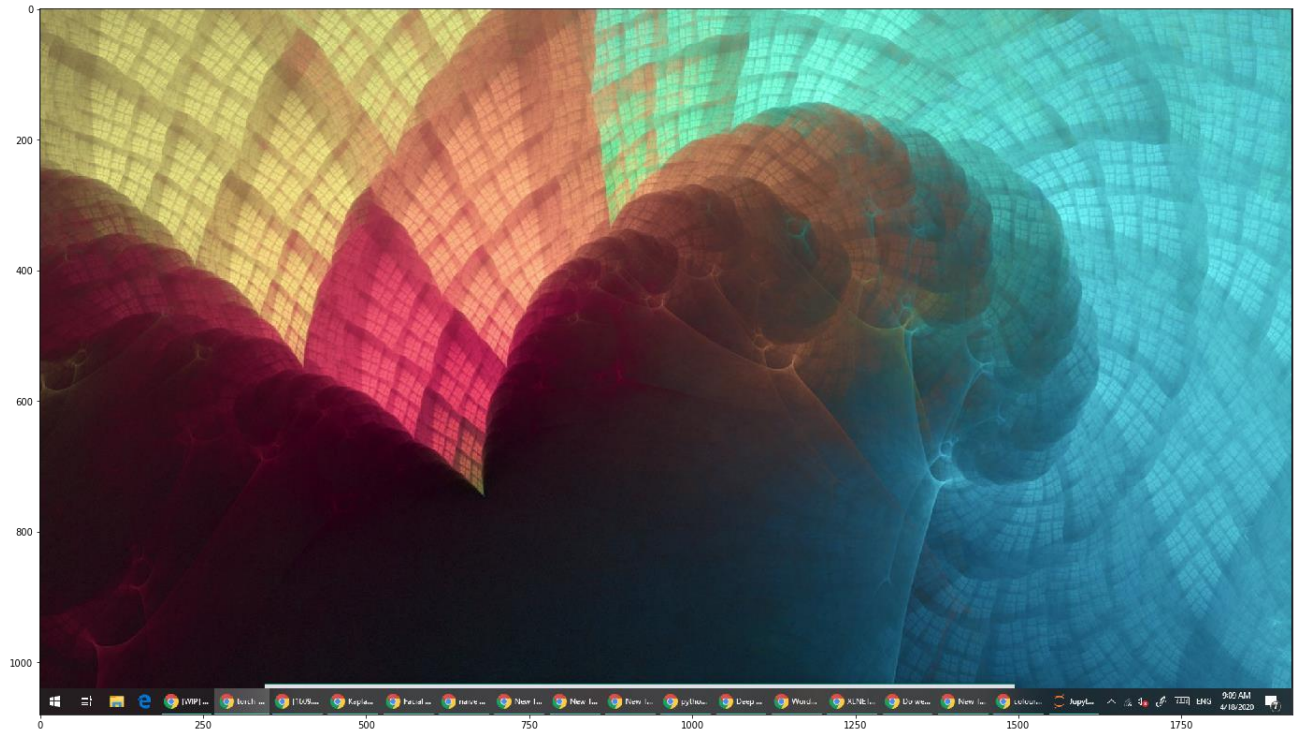
After code optimization



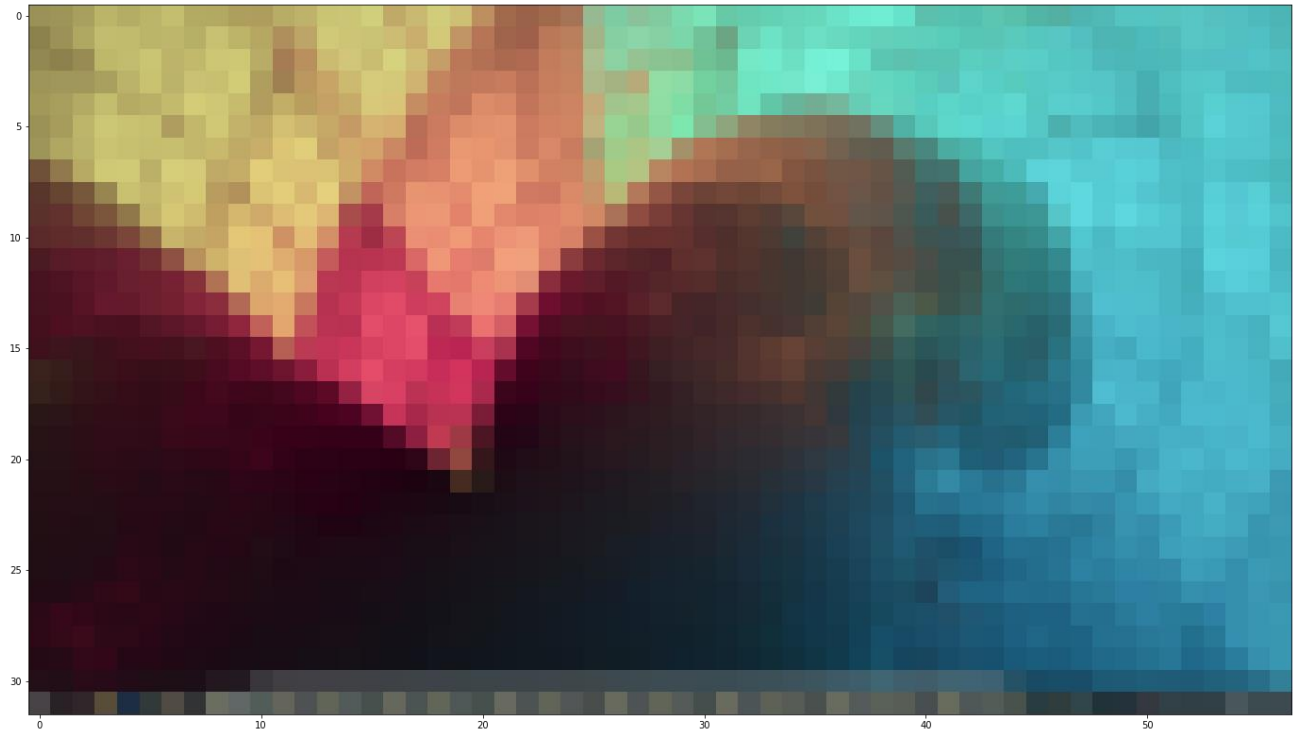
The output is reverted to NumPy, in float, then only use tensor for interpolation.
(use only tensor for calculation, GPU memory decreased significantly)

Finding the best down sampling algorithm (details can be referred to jupyter notebook)

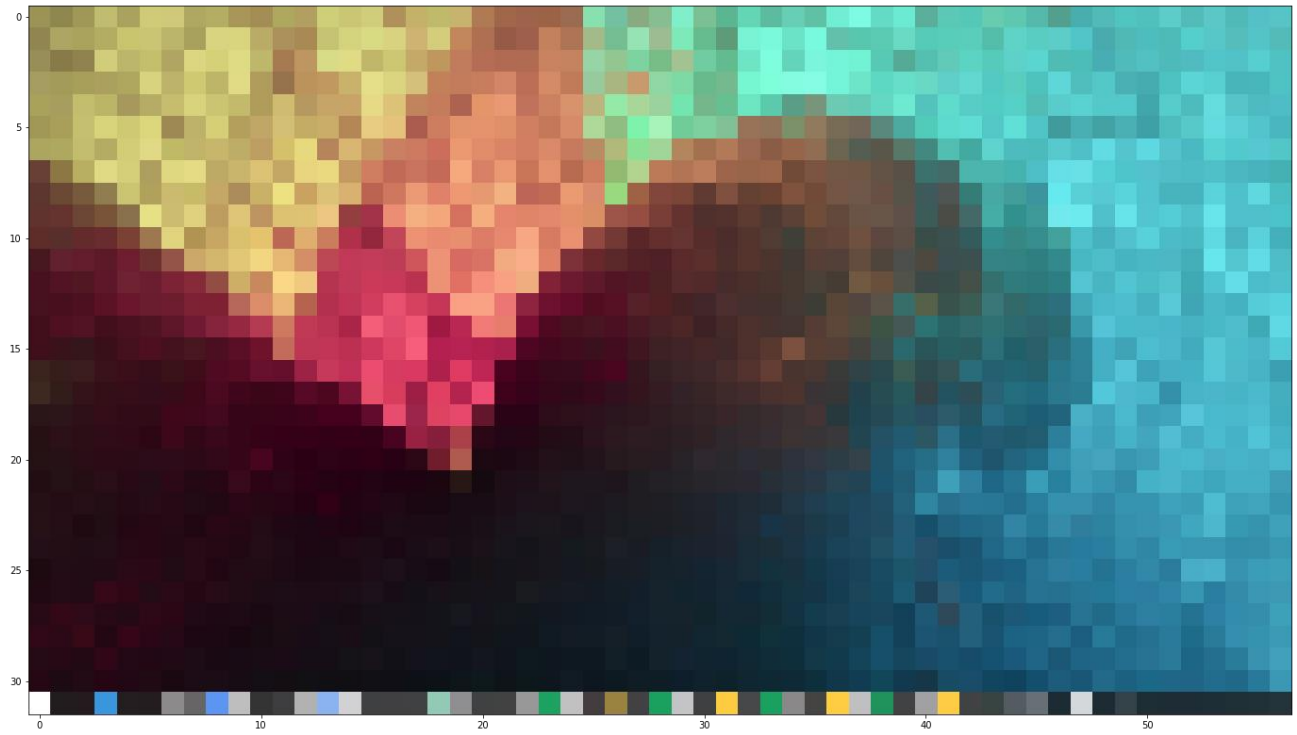
`torch.nn.functional.interpolate(`*input*, *size=None*, *scale_factor=None*, *mode='nearest'*, *align_corners=None*)



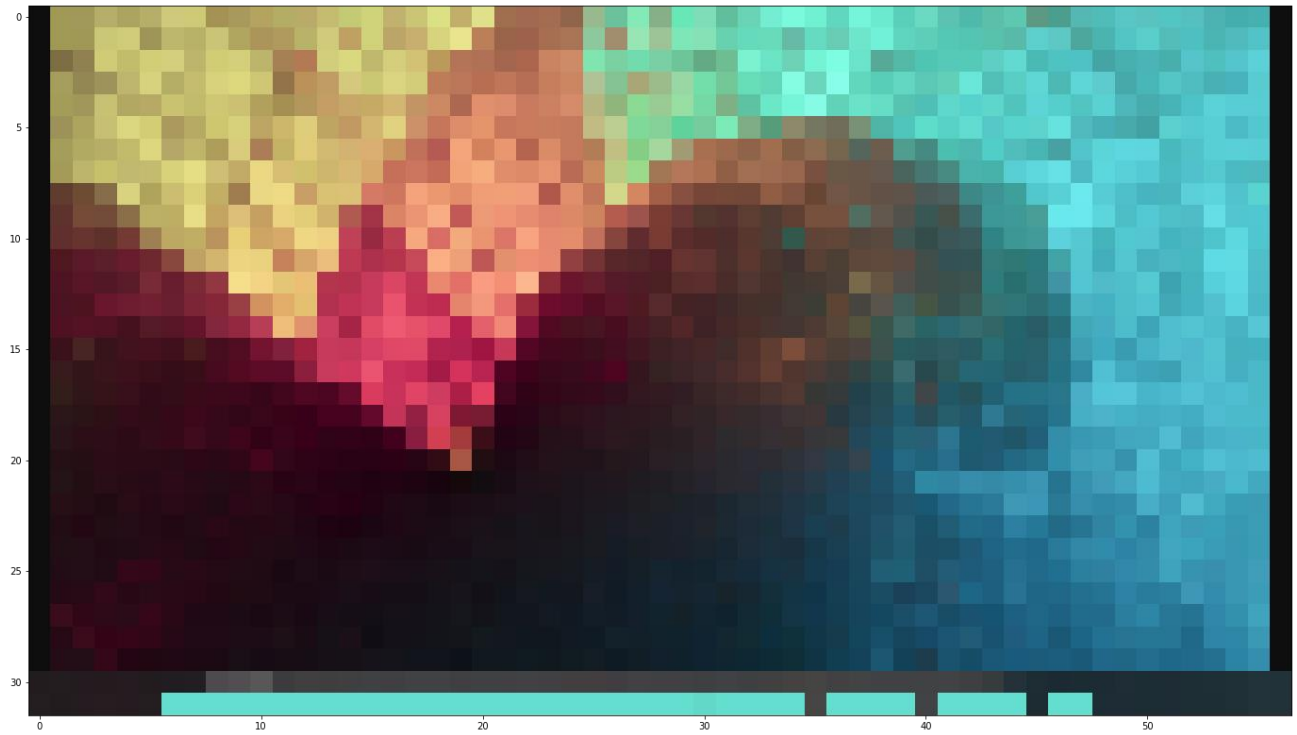
Original



'Area'



‘bilinear’



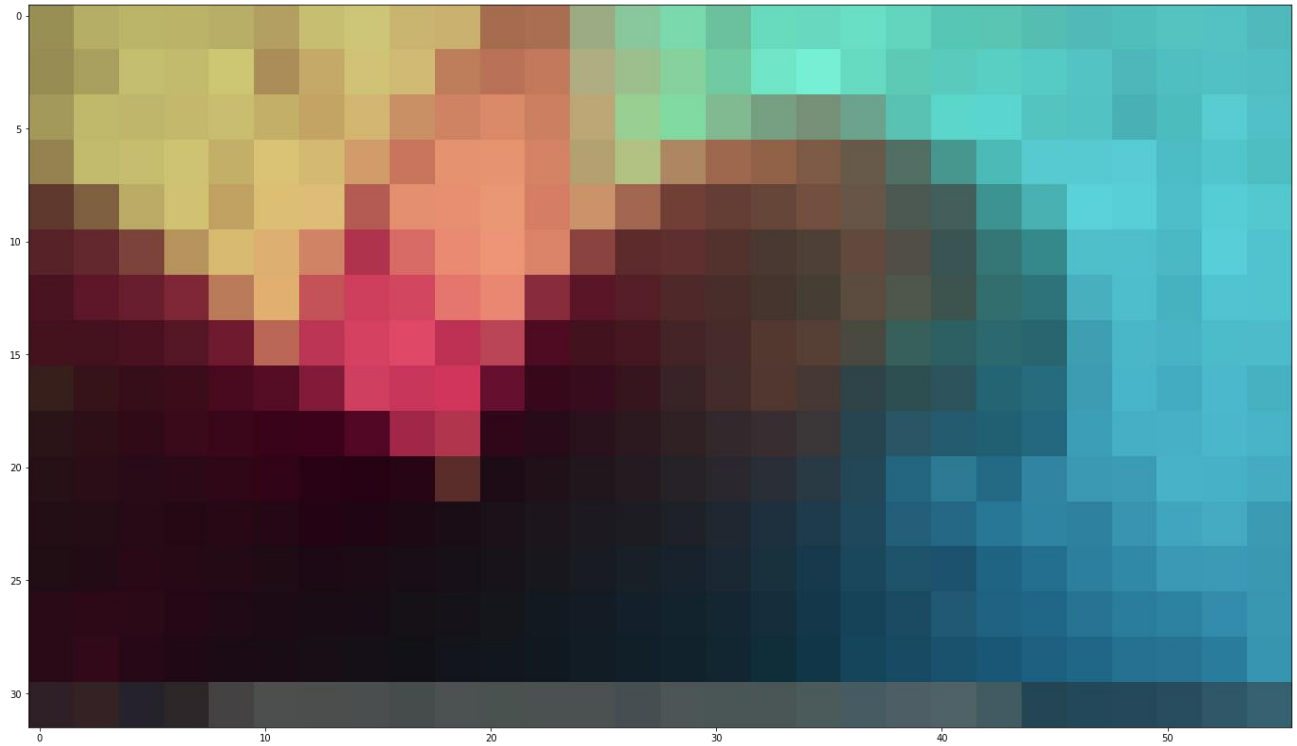
‘bilinear’ + align_corners=True

Hence it is best to use ‘area’, which gives a smoother feeling and is more generalized for the colors.

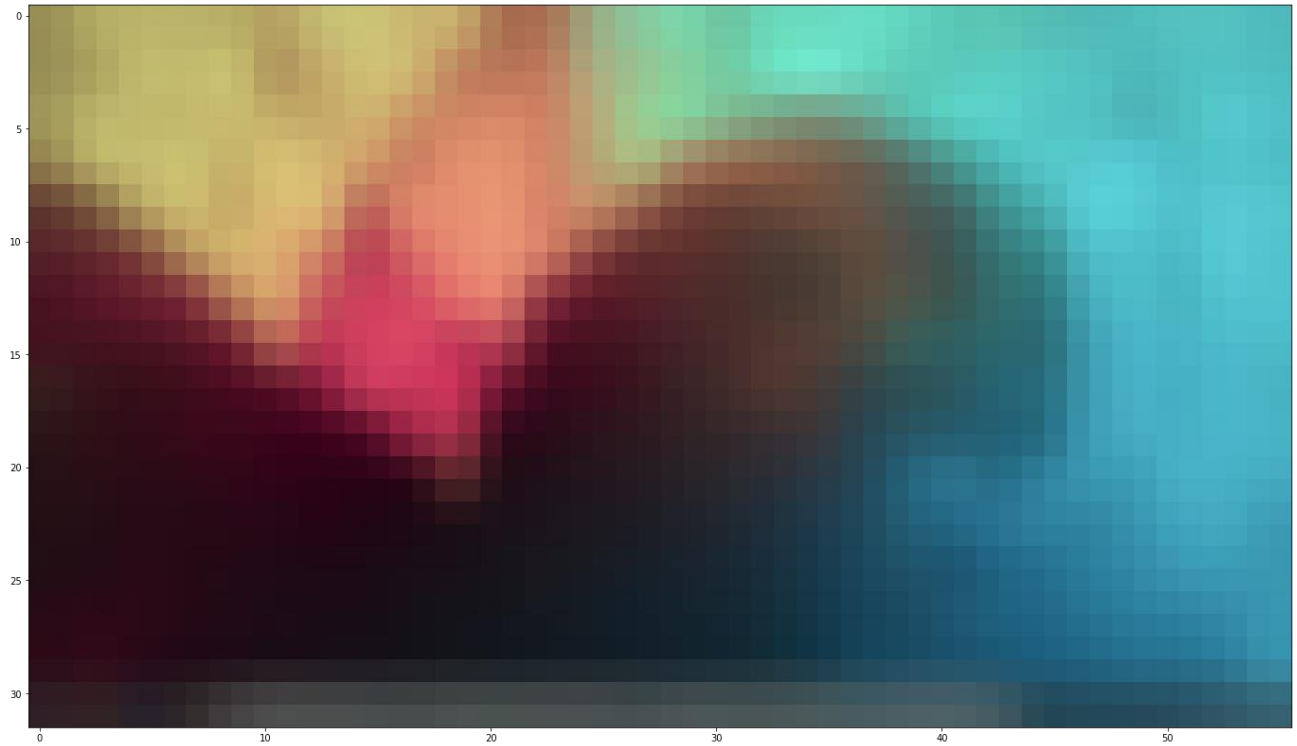
Downsampling and upsampling

Since the result is not smooth enough, I've decided to first down sample more and **up sample** it back to the original down sample size, in order to obtain a blur effect.

All are down sample by scale_factor/2 and upscale by 2, where scale_factor=0.03



'Area' upsampling, only double the size without considering neighboring cells.



'Bilinear' is ideal. Details can be referred from 'Find the best interpolation.pdf'.

Gaussian blur and Box blur

The tensor can be blurred by being convolved by a conv2d layer in Pytorch, since I'm not going to program the kernel, I'm using a tensor library called kornia to try the qualities and performances of gaussian blue and box blur. Details can be found in 'Benchmark on Gaussian or Box Blur.pdf'

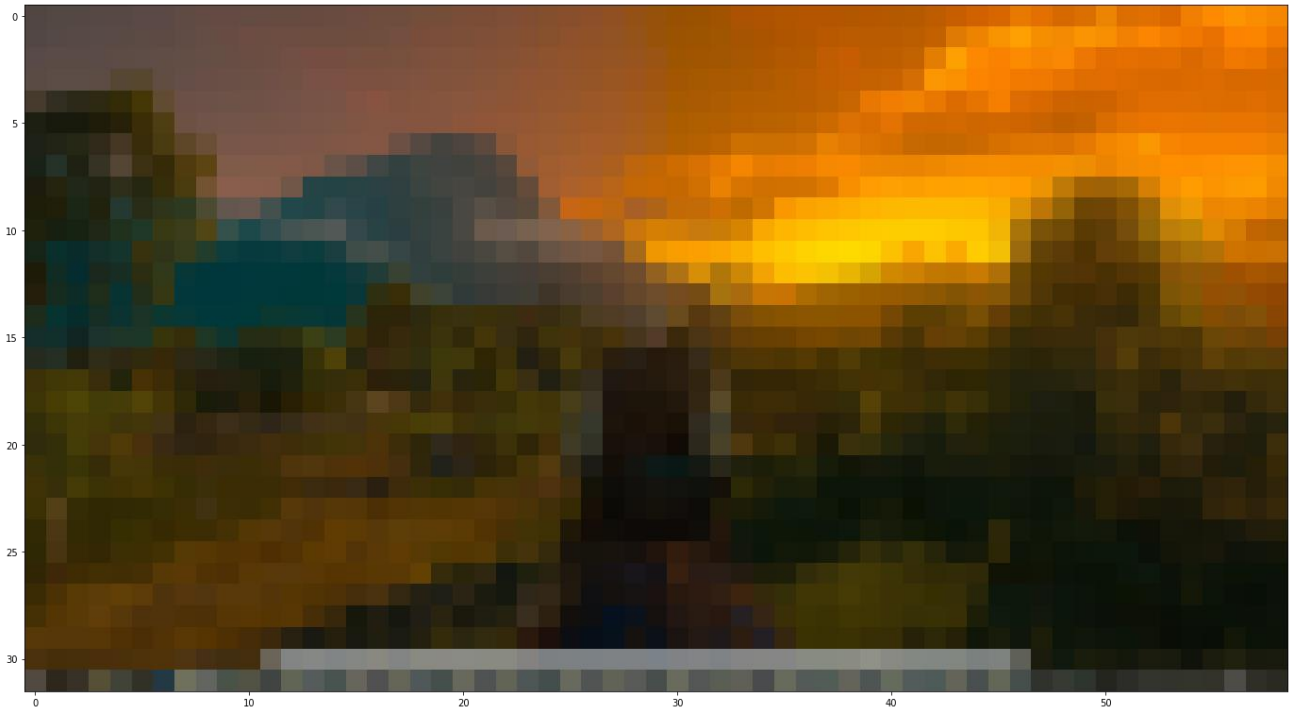
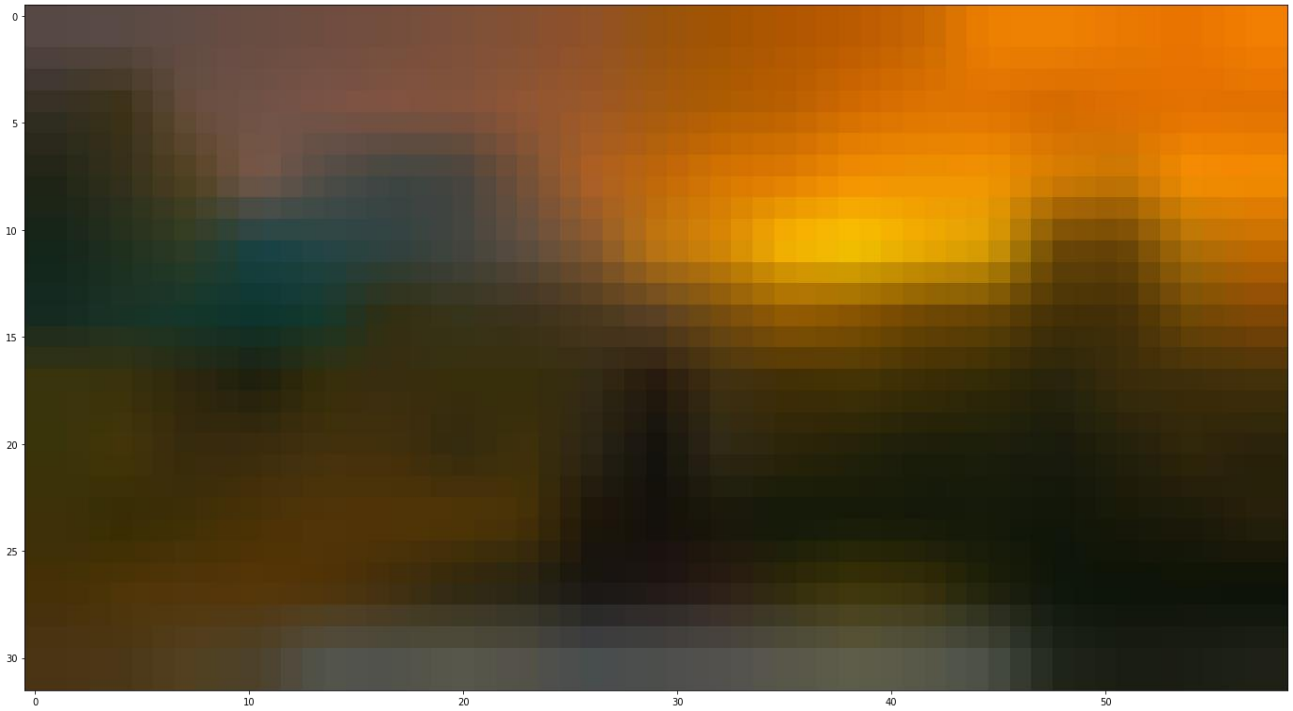
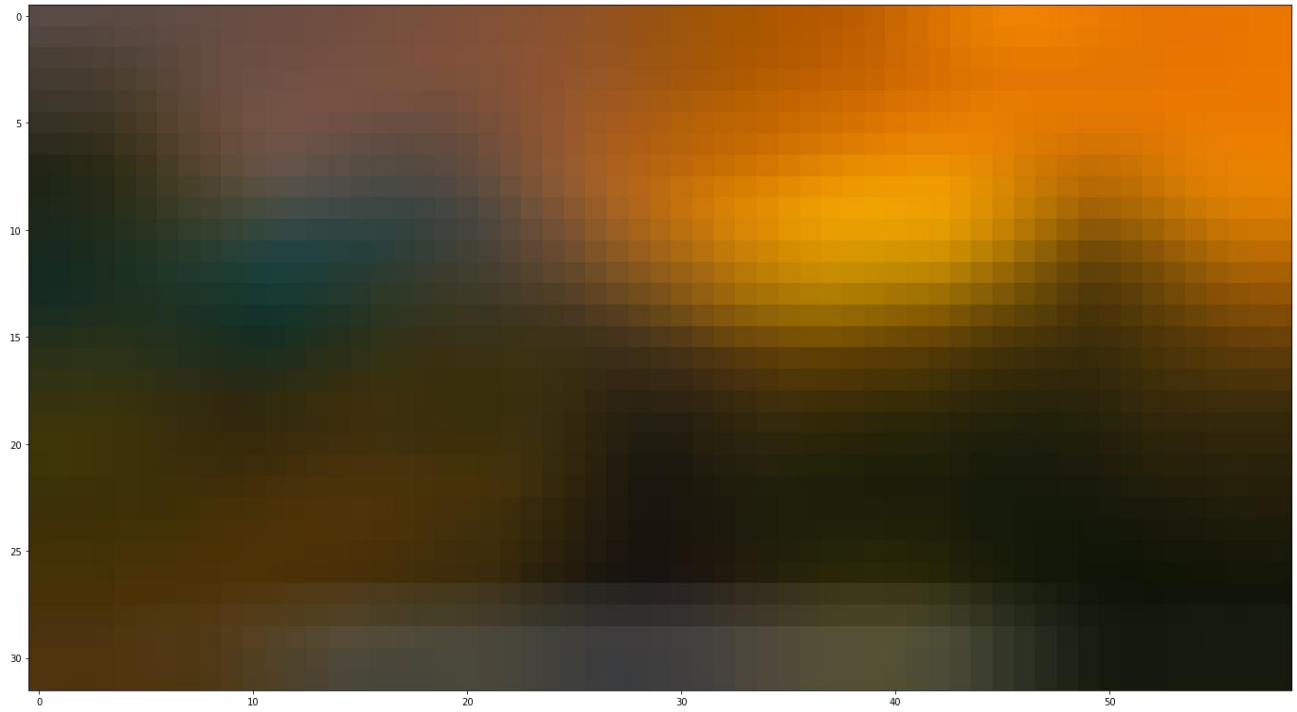


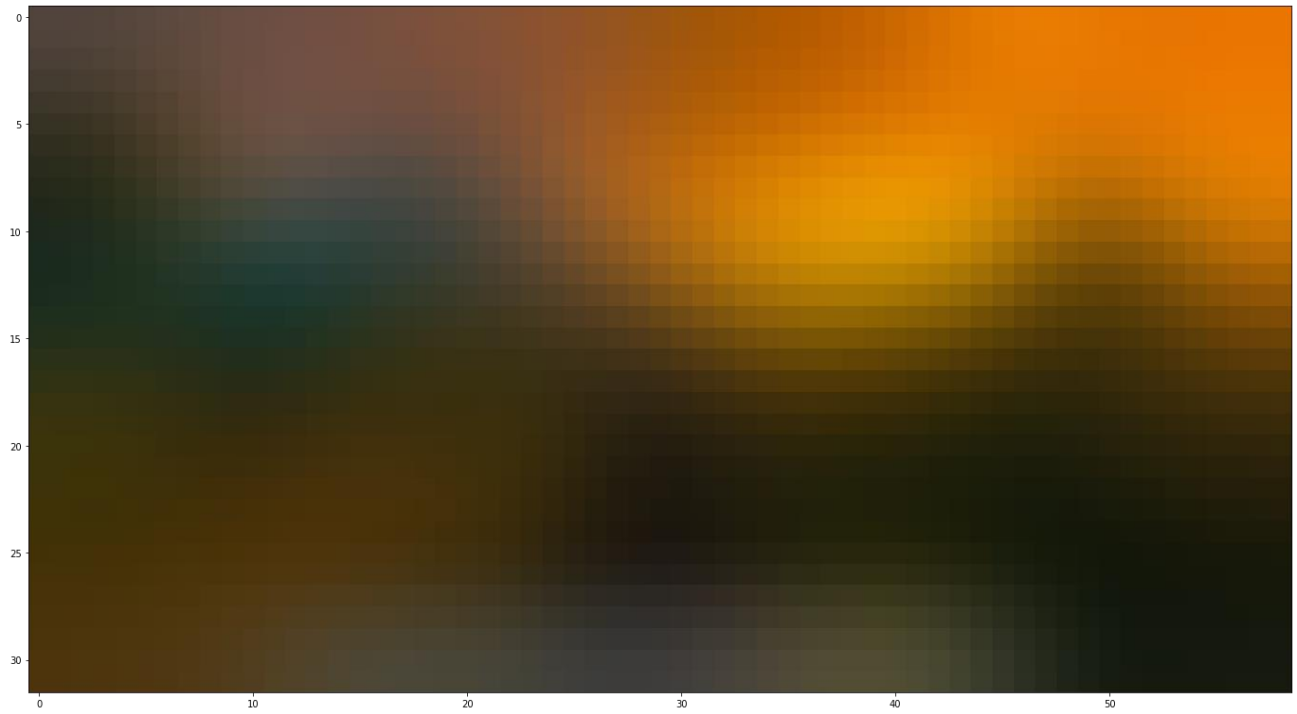
Image to be blurred: only down sampled to (32, 59)



Down & up sample: scale_factor=0.01 and up sample to (32, 59)



Box blur with kernel size [7, 7]



Gaussian blur with kernel size [11, 11] and sigma=3

They both look good, however, for a pre-transformed and processed tensor, it took the same time as converting a untransformed NumPy matrix to a tensor and interpolate twice and back to NumPy in my downupsample function. (~7 seconds for 60 operations)

Code optimization

It took me some time to realize for basic operation like indexing on a NumPy matrix, it had been better to do in NumPy. After some benchmarking and research on the Internet, it took some extra time for the computer to reference the location of the tensor in GPU memory, therefore, it's best to only do computation as tensor, but not basic operations. In order to optimize the code completely, I get rid of all the intermediate variables, and change the d3dshot output to NumPy float instead of Pytorch tensors.
The result is remarkable, the downupsample function took 7 seconds, and now it only takes around 1.5 seconds for 60 operations. And the python benchmark is not that accurate, for real use, I have nearly 40 fps in full HD, which is more than enough for watching videos. Details can be referred in 'Optimized.pdf'.

Code explanation

Python

There are two essential functions to be explained.

```
def down_up_sample(image_matrix, scale, size=(32, 59)):

# B = Batch size, C = Channel, H = Height, W = Width
```

The torch.nn.functional.interpolate only accept tensor in the dimension of (B, C, H, W), and the numpy matrix from d3dshot is in (H, W, C). I will have to rearrange the axis and then add a Batch dimension (size 1 in this case) in order to use this function. It took me some time to realize I missed the Batch dimension.

1. roll the dimensions of input matrix (H, W, C) -> (C, H, W)
2. add a dimension at the beginning as the batch size
2a. insert the Batch dimension at axis 0
3. create tensor from NumPy
4. down sample
4a. down sample by a scale factor, the lower the smoother
4b. interpolation mode, area with anti-aliasing
5. up sample
5a. up sample back to a fixed size
5b. Bilinear to interpolate between pixels, disable align corners
6. Copy back the tensor(use [0] to get rid of batch dimension) to cpu memory as NumPy
7. roll the dimensions back to (H, W, C)
7a. shift the first dimension (Channel) by three

I've managed to squeeze everything in one line, not sure if it's the right thing to do, but the performance has indeed been improved.

def image_to_signal(image_matrix):

This function is to extract the bottom, left, top and right border of an image matrix (H, W, C) and flatten into one dimension. Originally, I was using transpose and image flip, but that was not memory efficient since NumPy would copy the array, by pure indexing, it saves memory and gives a faster performance.

```
        image_matrix[-1, ::-1, :],  # bottom row, the last row and invert step ::-1
        image_matrix[-1:1:-1, 0, :],  # left column, exclude first and last row(included in top and bottom), invert
        np.resize(image_matrix[0], (55, 3)),  # top row, simple resize to 55
        image_matrix[1:-1, -1, :]  # right column, exclude first the last row in the last column
```

Only the top row is doing a simple resize. After the borders are selected, they are concatenated and flattened to one dimensional array in [r1,g1,b1,r2,g2,b2,r3,g3,b3…]

C++ in Arduino IDE

Normally, the webservers are placed in loop, however, for an async server, all the codes are done in the setup function. There is only the 'mood' mode breathing light is placed in the loop since it requires time from the ESP-32.

```
while (packet.available()) {
  colors[channel % 3] = packet.read();
  if (channel % 3 == 2) {
    leds[channel / 3] = CRGB(colors[0], colors[1], colors[2]);
  }
  channel++;
}
```

In this part of code, I've created a cache (colors) of size 3 to cache the color channel of each LED in the one-dimensional array. And for every final channel (mod 3 = 2), it's going to apply the color cache to the respective led number (divided by 3)

The rest of the functions are straightforward. One of the things worth mentioning is that the color hex can be represented by decimal, so I've always converted them to uint32_t in case of confusion. The post request is expecting decimal integers, not hex, and the web application is also converting the selected hex to decimal.

jQuery

For the web application part, I've used Bootstrap 4 to create the front-end. And I used jQuery ajax to handle all the requests.

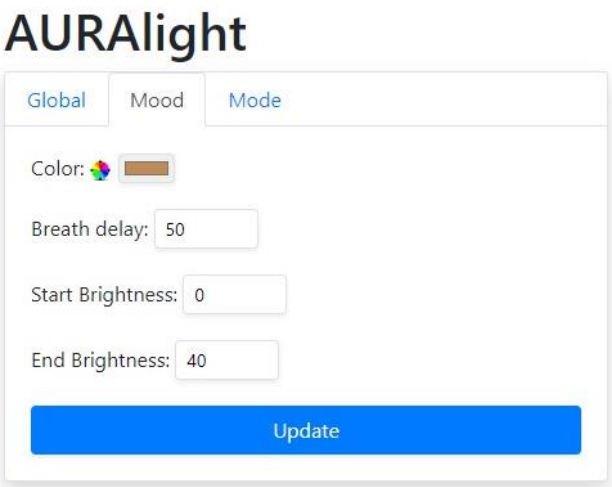
Reflection

At the beginning, I thought it was going to be quite an easy, but it turns out it's totally not what I've imagined it to be. At first, when choosing the led lights, I spend over two weeks in order to find the best one for my project, which I aimed for a cost-effective solution. I bought the WS2812BECO version, but I misread the description and hence I have double the number of LEDs.

The second step is to decide how to apply it on my monitor. I was foolish to think that I should wrap it around the edges, and that was what I had planned. Then I realized I want to light to be cast on the wall, not on my table. I had to wrap it around and kept the corners protruded so that I can save the cutting and soldering part.

At the end, I am quite satisfactory of my optimization. Thanks to Pytorch, it was much faster than the programs written in compiled language. However, the memory usage of my program is concerning. Since Pytorch require lots of 'base' memory to run. Whenever there's some tensor calculation, the CPU memory is going to increase up to 2GB. Luckily, the memory keeps at around 500MB if the current screen is still.

However, there was something buggy with SPIFFS. I had to upload the web application to a SPIFFS partition in ESP-32, but sometimes the partition becomes corrupted. Hence, I had to leave it and use the web application on my computer. Still, it's pretty flexible since it is also a restful server.



A simple Bootstrap 4+jQuery web app

Future works

I'm going to implement this as part of the visual aid of my Final Year Project. I'm going to use pretrained natural language model like XLNet to do sentiment analysis of a book and create corresponding light visualization as the sentences are spoken by my trained TTS model.

Another interesting idea is that, by analyzing the subtitle, the brightness can change according to how intense it is, where it can be measured by amplitude and sentiment analysis again. Thus, I can enjoy a more lively performance.

Video

https://youtu.be/IXO1Iv0GNnc