Challenges to convert Super Resolution GAN

Omar Faruk Riyad
1610634042
North South University

Arif Suhan
1610437042
North South University

May 2, 2019

Abstract

Deep learning approaches to single image super-resolution have achieved impressive results in terms of traditional error measures and perceptual quality. We are focused on the area where we could reduce the load over the network and run on the browser. To achieve this, we have faced a lot of challenges like loading the weight of pre-train model or running the network model in the browser. As we are using pure vanilla javascript, we also have faced a lack of libraries. When it's time to load weight and compute the network function, the browser cache crossed its a limitation. For solving these challenges, many solutions are crossed through our mind. After so many testing, we take the best solutions. Still, some challenges are not solved.

1 Introduction

Since the dawn of the internet, reducing network load while transferring data has been the great challenge of computer science. As the most transferred data over the network are image data, if we could get high-resolution images from the low resolution that could save the network load by a significant amount. We wanted to try out a working mechanism that could achieve this particular goal while leaving the heavy duty on the user's computer. The state-of-the-art architecture ProSR is one of the best ways to retain image information to convert a low-resolution image to high resolution. The mechanism that we tried was to perform the computation on the browser side. While the data sent by the server is low-resolution image later it will be generated as high resolution. To make a dense model work in the browser is a challenging task. Along with the dense layer they have also used bicubic interpolation layer by layer to make the high resolution even better for this particular model. The sole purpose of choosing a complex network is exploration the compatibility issues over variations of interfaces, in this case, pure JavaScript without any libraries.

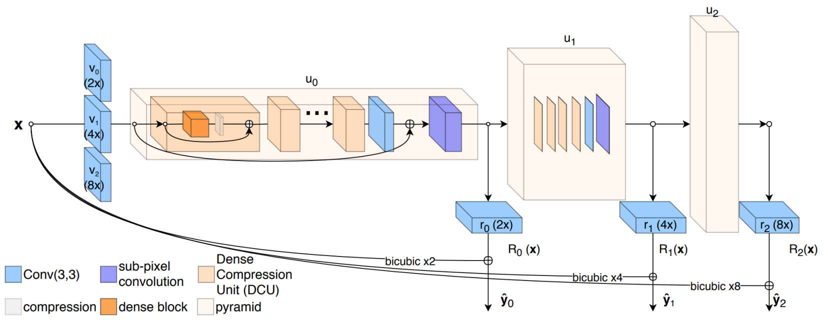
2 Model Description

2.1 DenseNet

Dense networks are the next step in increasing the profundities of deep convolutional networks. When CNNs go deeper, the problems arise. This is because the path from the input layer for information becomes so large that it can disappear before reaching the other part of the output layer (and the gradient is in the opposite direction). DenseNets simplify the pattern of connectivity among layers in other architectures.

2.2 ProSR GAN

Generative adversarial networks (GANs) have emerged as a powerful method to enhance the perceptual quality of the upsampled images in SISR.The model ProSR contained a dense layer and a dense compression unit. The model also used bicubic interpolation and addition with some of the layer output, for which it is one of the expensive model to perform computation with. But the result produced by this ProSR is amazing interms of quality of produced the image. [1]



3 Challenges

3.1 Converting Model

The first problem we got in order to get further with the idea is to convert the model to JavaScript natively. We tried out a bunch of different methods to convert the model but because of the model contained dense layer, it was really difficult to put into code since the sequence of the network has to be followed. We found that we could easily convert PyTorch model into an Open Neural Network Exchange (ONNX) model, from the base of ONNX it performs operations in Pure Basic Language. We extracted the Pure Basic Model code from the ONNX model, from which we found the underlying repeated network in the right sequence of its weights. We also found that there are 610 layers in the neural network and has about 7 operations which also have multiple different parameters. The pure Basic language is similar to native JavaScript, by performing some regex operation the code, we easily converted it to a native JavaScript code.

This was the easiest challenges that we had to face over the entire time working with this project.

3.2 Weight to JSON

The next challenge that we had to face is to convert weights and load it to the browser. Converting the model weights to JSON was complex but it was easily done. Here again, we used the ONNX model to get the weights from since our model network was generated from the same model. The model had all weights in the sequence that we needed it to be thus parsing the weights wasn't difficult. At first, we worked with float32 data to have better precisions but the loading time was taking more than what we expected. Also, the file size has gotten about 345MB even after minifying the JSON file. It was a challenge to fix because we had to lower the file size and make it work at the same time because the browser couldn't cope up with the file. We solved this issue by reducing the weights from float32 to float16 which required fewer data and less memory size. It was not an easy thing to do but we had to choose float16 because of our operations were running in the CPU. We made work on the cost of losing precisions.

3.3 Engine Compatibility

At first, we were using Google Chrome to run our tests. But Google chrome runs on the chromev8 engine which we couldn't make it load the weights JSON faster. On the other hand, the firefox uses spidermonkey engine that makes faster to load the weights. Also most of the time, Google Chrome used to crush where firefox didn't.In the comparison of javascript engine compatibility, spidermonkey is better than chromev8 especially for loading JSON and garbage collection purpose. As Spidermonkey compiles any javascript code to bytecode, it has an auto garbage collector. But chromev8 hasn't and it compiles the code to machine code.

3.4 Performing Operations Issue

Most of the users of the browser does not have GPU support since the whole idea of exploration was CPU based. Our JavaScript engine compiles the javascript code to byte code rather than machine code. So we lose some of the performance because of it.

3.5 Computation Issue

Computation with the model was higher than we expected. Since it had a huge amount of weight and layers to it and the model used bicubic and previous layer output farther down the network. It had to store a lot of data to the memory. We tried some of the low-resolution images to convert them to high resolution but we failed. Also to finish computing of the full network model, we need a lot of time for execution.

4 What to expect

If we could solve all these problems, we could have faster data transmission for which the network could be reduced to more than 25% by assumption. This method is new and diversified to our current standards. We didn't tired training on JavaScript natively because that wasn't our sole purpose. But if we tried to write the model and retrain it we could achieve better result. However, that is out of the scope of what we were trying to achieve.

5 Evalution

5.1 Run Time

We run our model regarding other errors on the Firefox engine to see if it perform anyhow. But on runtime, we have seen that the model proSR usage 28GB of memory to perform computation, which is beyond anyone would expect. If we could maintain the memory better way we could achieve a bit better performance.

6 Conclusion

In our exploration, we have seen that computing in the browser is highly costly to run operation. With an efficient library, we hope this problem can be solved. Also if we can use GPU for computing through the browser, the performance with increase a lot and save our time but most of the users don't have a GPU to begin with. Finally, it can be said that the idea of using deep learning in the browser could open up new opportunities in the realm of computer vision problems.

References

[1] Brian McWilliams Yifan Wang, Federico Perazzi. *A Fully Progressive Approach to Single-Image Super-Resolution.* CVPR, 2018.