CSCI 1430 Final Project Report: Beyond Super Resolution: An adversarial analysis and extended application of ProSR

SuperZoom: Mary Dong, Peter Huson, Michael Litt, Gabe Weedon
Brown University
10th May 2019

Abstract

Super-resolution is the process of upsampling the resolution of an image. Although the research community has put significant effort into developing robust super resolution models, there has been little adversarial testing and analyses performed on most architectures. In this paper, we conduct an adversarial test on a top-performing architecture called ProSR, with hopes of understanding the structure and vulnerability of this model on a deeper level. We also investigated novel applications of ProSR, which was built to upsample single images. Our project helped us gain an intuitive understanding of deep-learning based super-resolution and pushed us to consider future directions in super-resolution research.

1. Introduction

Single-Image Super-Resolution (SISR) has been a hot area of study in recent years. The ability to turn a low-resolution (LR) image to a high-resolution (HR) one is useful in many specialized domains (e.g. medical imaging, satellite imaging) and has widespread commercial potential (e.g. high quality video streaming).

Every year, NTIRE hosts a competition for top-performing SISR models. We realize it's unrealistic for us to build a model that competes with the prize-winners, but we can stress-test and build on top of the current state-of-the-art. Here we focus on the ProSR architecture which ranked 2nd in terms of SSIM and 4th in terms of PSNR in the NTIRE 2018 SISR challenge (Team Name: DRZ) [3]. Speed was important to us as we wanted to test the model on a large number of images—ProSR stood out in this aspect because it ran five times faster than other top-performing models.

In this paper, we first conduct an adversarial testing of ProSR (and its extension, ProGANSR) using a variety of images, patterns, and textures. We then move on to a repeated stress-test that sheds light on what ProSR/ProGANSR is particularly sensitive to. We also apply the ProSR model to other tasks: First we extend the SISR framework to upsample the resolution of a video; finally, we use the model to upsample a video in a new dimension—time—as a means of motion interpolation.

2. Related Work

Historic approaches to SISR have enjoyed limited success due to the complex and ill-defined nature of mapping LR inputs to HR outputs. Recent deep learning approaches, however, have drastically improved performance on this task. CNN and GAN architectures have been especially successful. Yang et al. [5] summarizes the general trends in SISR in their March 2019 paper. We won't go into specifics here, but it's important to note that SISR models are evaluated based on peak signal-to-noise ratio (PSNR), structural similarity index (SSIM), and efficiency.

In our project, we focused on an architecture called ProSR by Wang et al. [4]. This model has an asymmetric pyramidal structure with more dense layer blocks in the lower levels of the model. The original ProSR is a CNN, while the GAN version (ProGANSR) features a generator and discriminator. Wang et al. took two novel approaches in their model. First is progressive reconstruction, which upsamples LR images in small steps to achieve a larger overall upsampling factor (for instance, to get a $\times 8$ upsampling factor, we would do $\times 2$ upsampling three times). Second is curriculum learning. By training the model progressively ($\times 2$ first, then $\times 4$, ...) and gradually introducing new layers, Wang et al. were able to decrease training time and produce more stable GANs.

In terms of adversarial testing, there has not been robust adversarial testing performed on ProSR. For the application component, current motion interpolation techniques are generally not deep learning-based and instead focus on estimating optical flow vectors [2]. On this front, we took a more exploratory approach; we were curious to see whether a ProSR model trained to upsample resolution could also perform temporal upsampling.

3. Methods

3.1. Methods: Adversarial Testing

We used two different methods to determine whether there were specific cases where the model did not behave as expected.

3.1.1 Interesting Images

In order to test the models, we ran a wide variety of images through the neural network in attempt to find cases where the models produced results that seemed flawed. Since there is no metric to evaluate the accuracy of an upsampled image, we evaluated the upsampling based on how reasonable the upsampled images look to the human eye. We tested on a wide range of images, including standard photographs, camera calibration images, and pixel art to see how the network would analyze the different textures and patterns.

3.1.2 Repeated Stress Testing

As a form of stress testing of the network, we continually upsampled images with the network and ran the result (downsampled to its original size) back through the network. We did this for 1000 iterations on both ProGANSR and ProSR.

3.2. Methods: Applications

In addition to stress testing, we also applied the ProSR to other tasks. We went beyond a single-image framework and used the model to upsample a video. Going even further, we also wanted to see if ProSR, which is trained on upsampling pixel resolution, could perform well in another dimension: Time. We used ProSR to make a low-framerate video smoother via frame upsampling and motion interpolation.

Here we use a combination of our own code (snippet below), ProSR, and FFmpeg, a robust video-manipulation software [1].

3.2.1 Video Resolution Upsampling

We took a video of a street and selected a cut with lots of movement (cars, pedestrians, etc). We turned the video into frames of images (25 FPS) using the command `ffmpeg [INPUT VIDEO] -r 25 [OUTPUT]`. We downsampled these images by reducing each one's height and width by a factor of 8 using the FFmpeg command `ffmpeg -i [INPUT] -vf "scale=iw*.125:ih*.125 [OUTPUT]`.

Next we ran each frame through the ProSR model and turned the directory of upsampled images back into a video using `ffmpeg -framerate 25 -pattern_type glob -i [INPUT] [OUTPUT]`.

3.2.2 Frame Interpolation

In order to upsample in time, we first stacked the frames of the image on top of each other to create a 3D matrix. We then sliced the matrix across the time dimension so that we get a single row of pixels from each frame (code snippet shown below). Next, we upsampled these time-slices [1] using ProSR and re-stacked them together. We sliced our upsampled 3D chunk across the original dimension to get an upsampled series of frames (if we originally had 20 frames and did a $\times 2$ upsampling across the time slices, we would end up with 40 frames). Finally, we used FFmpeg to turn these new frames into a new high-framerate video.

We had to perform motion interpolation on greyscale images since we wanted 2D frames with only one channel.

```
all_imgs = []
for filename in directory:
        img = cv2.imread(filename, 0)
        all_imgs.append(img)

dim1 = all_imgs[0].shape[0]
dim2 = all_imgs[0].shape[1]
stack = np.array(all_imgs).reshape((len
    (all_imgs), dim1, dim2))

for i in range(dim1):
        # uncomment one of the blocks
            below
        time_slice = stack[:, :, i]
        # time_slice = stack[:, i, :]
        img = Image.fromarray(
            time_slice, 'L')
```

Figure 1. Two time-slices. The height of each slice is equal to number of frames in the video. Since we took a low-frame video with only 22 frames, the slices were quite thin. Each 2D frame contributes one row of pixels in each slice.

4. Results

4.1. Results: Adversarial Testing

4.1.1 Adversarial Images

Though we tested the network on a large number of images, we only included the most interesting and unexpected ones here. We present four examples featuring: text [2], noise [3], alternating black and white pixels [4], and pixel art [5].

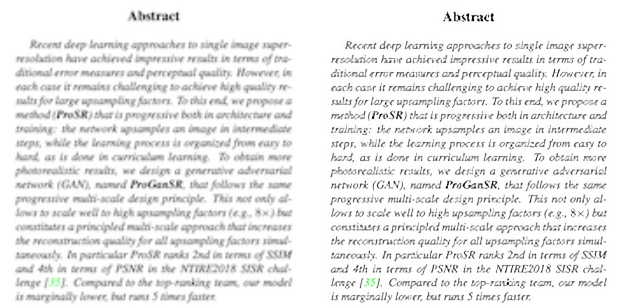

Figure 2. An example of text, downsampled 4x, and then upsampled 4x with the network. *Left:* Downsampled. *Right:* Upsampled.

For the text example, the letters were made much sharper and the blurriness from the downsampling is entirely removed. The original shapes of the letters were not fully recovered; however, there is definitely an overall improvement in readability.

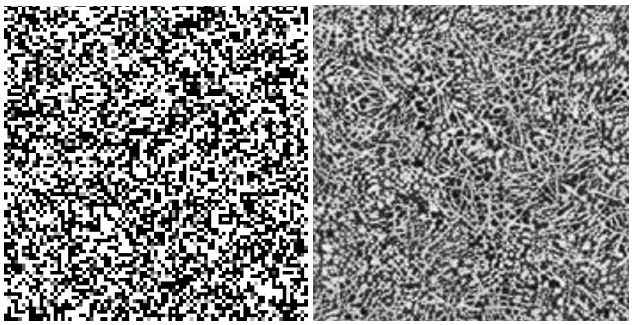

Figure 3. Noise. *Left:* Original. *Right:* Upsampled.

When upsampling noise, the model finds patterns that were not there in the first place, and hallucinates distinctive 'strands' from groups of pixels.

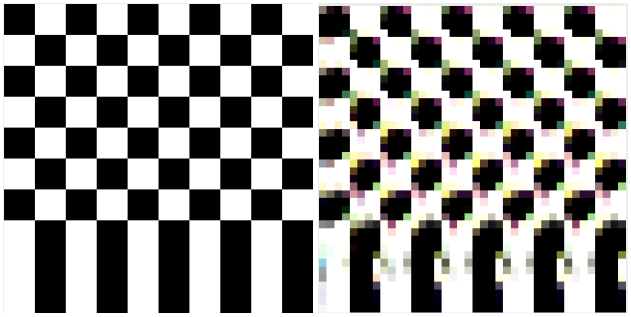

Figure 4. Alternating pixels. *Left:* Original pixel image, each black and white square is an individual pixel. *Right:* 8x upsampled version. Images enlarged to show effect.

For the alternating pixel example, the network added very pronounced colors around the completely black pixels. Rather than keeping the pixels as solid black squares, it made the edges softer and added a different color to each edge of the pixel.

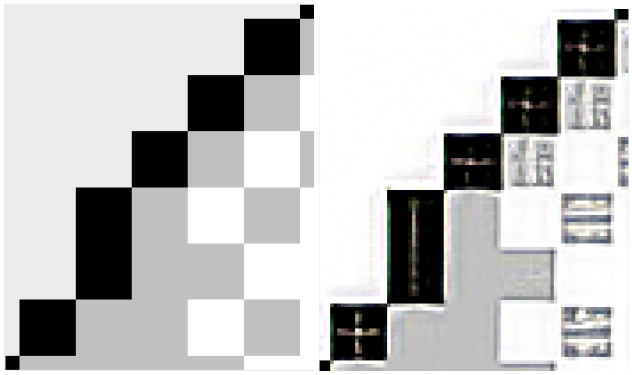

Figure 5. Cropped section of a grayscale pixel art image. *Left:* Original. *Right:* Upsampled 8x result.

In the pixel art example, the network hallucinated color around the edges of pixels even though there was no color in the original image. It also added 'panes' separating individual pixels of the same color, which made the upsampled image look less like pixel art than the ground truth. In addition, it added an outline that provided more dimensionality than the 2D image originally had.

4.1.2 Repeated Stress Testing

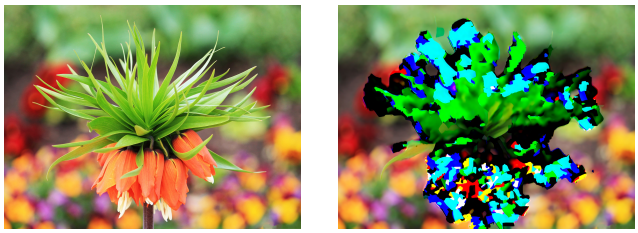

Figure 6. Results of stress test. *Left:* Original. *Right:* Image run through network 1000 times.

Much of the underlying data in the image ended up being lost and replaced with patches of color. ProSR expanded these patches of color in the edges, indicating that the network is particularly sensitive to color and constract changes.

4.2. Results: Applications

4.2.1 Video Resolution Upsampling

A sample of two frames is attached below [7] and the full video demo is included in the code submission. The results here are what we expected: the output was visibly sharper than the input, especially along edges with stark contrast changes. For instance, the white car got a lot sharper in contrast to the dark background and windows. However, the video was still blurry and cartoonish in areas with lots of noise (e.g. tree branches, street signs).

Figure 7. *Left:* LR frame *Right:* $\times 8$ upsampled HR frame. Image cropped then enlarged to show effect.

4.2.2 Frame Interpolation

Though we didn't go into this with high expectations, we were surprised to find that our technique actually produced overall smoother videos. We've included the video demos in the code submission, and below we show a frame before and after upsampling $\times 4$ [8]. It was interesting to see that ProSR successfully picked up on motion in the video and seemed to only hallucinate new information in regions with lots of movement (around cars, pedestrians).

Figure 8. *Left:* Low FPS frame *Right:* $\times 4$ upsampled high FPS frame. The model hallucinates information around moving objects. On the left, we can see the shape of a hood hallucinated in the area directly in front of the car.

4.3. Discussion

4.3.1 Discussion: Adversarial Testing

We found from our adversarial testing that the network does not match the ground truth when given an artificially produced image. In the noise, alternating pixel, and pixel art examples, the network hallucinated patterns that never existed in the ground truth images. This behavior is unsurprising, since the network was trained on photographs; however, it does shed some insight into how the network operates.

The output from upsampling the noise 3 demonstrated that the network attempts to reconstruct lines that are lost in low resolution images. Another example of this can be found in the pixel art example 5 where the corners of the black pixels are blended together.

Adding panes between the pixels in the pixel art shows that the network attempts to reconstruct other features that are lost when an image is pixelated.

Also, the adding of color to the edges of pixel is likely a result of the network interpolating the RGB values between pixels. This is clear in the black and white pixel example—there is a stark difference between the pixel values, so the network is adding color values that are in between pure black and pure white at the boundaries of the pixels. This phenomenon is taken to the extreme in the stress testing example, where the colors are added around edges. Eventually, we end up with bright patches of pure cyan, red, and yellow; this indicates that the RGB values of pixels eventually to go 0 or 1.

4.3.2 Discussion: Applications

The result of the video resolution upsampling was expected. Since we operate on the video as a collection of 2D images, the overall effect of upsampling a video with ProSR is consistent with the model's performance on singular images.

The result of frame interpolation, however, was quite surprising. Even though that ProSR model was trained on LR images rather than time-slices, it performed decently well as a frame-interpolater. This is perhaps because ProSR has a smoothing effect on images: There are discontinuities in regions with movement in low FPS videos that show up as un-smooth regions in time slices. ProSR, which upsamples by smoothing textures and adding contrast around edges, is thus able to fill in details in these regions. The result is a higher framerate video with regions of motion that appear smoother in the output than in the choppy input.

One caveat of these applications is run-time. Since upsampling each frame by $\times 4$ takes around two seconds, upsampling an entire video is a time-consuming process (in the frame interpolation case, this took around an hour to run). Going forward, it would be worthwhile to explore ways to speed up video upsampling, with hopes of being able to improve resolution and FPS in real time.

5. Conclusion

Through our time working with the ProSR architecture, we were able to determine its strengths and weaknesses, as well as find new applications for the network. The network is best suited for improving the resolution of photographic images, and is particularly good at enhancing edges that are lost in low resolution images. It does not perform as well on digitally created images, where it ends up hallucinating details that were never part of the ground truth image.

Additionally, we were able to apply the network to increasing the resolution of an entire video and also adding frame interpolation.

Going forward, a point of focus for SISR researchers is creating models that only hallucinate information when needed (i.e. it's able to distinguish a LR input from an already-HR one). Of course, extending SISR to videos will also be a challenge, but we've shown that this approach has a lot of potential. Perhaps having multiple LR frames of a scene will actually yield more useful information that can be used to construct more accurate HR frames.

References

[1] F. Developers. ffmpeg tool (version be1d324) [software]. 2016. 2

[2] S. Niklaus and F. Liu. Context-aware synthesis for video frame interpolation. *CVPR 2018*, 2018. 1

[3] R. Timofte, S. Gu, J. Wu, L. V. Gool, L. Zhang, M.-H. Yang, et al. Ntire 2018 challenge on single image super-resolution: Methods and results. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018. 1

[4] Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. Schroers2. A fully progressive approach to single-image super-resolution. 2018. 1

[5] W. Yang, X. Zhang, Y. Tian, W. Wang, J.-H. Xue, and Q. Liao. Deep learning for single image super-resolution: A brief review. 2019. 1

Appendix

Team contributions

Mary Dong Mary made the pipelines for the video applications. She was in charge of creating and tweaking the workflow for both video resolution upsampling and motion interpolation. She analyzed the results of those and was involved in writing up the team's results.

Peter Huson Peter set up a VM instance using Google Cloud platform that enabled us to test the ProSR network. He installed the proSR repository and performed testing of the network on the DIV2K dataset, which is the dataset that the network was trained on, as well as some of the adversarial image testing. Peter also took the video of Waterman street that was used in the 'Applications' section.

Michael Litt Michael was in charge of finding images for the adversarial testing and running them through the network. He analyzed the results of the adversarial testing and theorized why some of the bizarre results turned out the way they did. He was also involved in writing up the team's results.

Gabe Weedon Gabe was in charge of the stress testing portion of the project, and wrote the script that repeatedly put an image through the network and upsampled it. He ran this script on images for both the ProSR and ProGANSR architectures. The ProSR result is the one included in this paper.

5.1. Submitted Code Files

Our project wasn't coding-heavy. We tried to leverage image/video processing software such as FFmpeg, and instead focused on coming up with interesting adversarial test cases, understanding why certain phenomenon occurred, and testing different application approaches. We did write code to help with preprocessing and stress testing.

repeat.py - Used for repeated stress testing; upsampled an image using the network and downsampled it back down to its original size to be put back into the network for a set number of iterations.

img_preprocessing.py - Used to preprocess each frame of a video. `get_time_slices` stacks frames and creates a series of time slices. `crop_all_center` crops a specified region of all images within a directory (used to create LR samples from HR videos).

main.py - Run's functions in **img_preprocessing.py**, given a specific path. Everything is commented out right now because it was meant to be run once as a preprocessing script.

5.2. Video Demos

res_before_after.mp4 - *Left:* Original low-resolution video. *Right:* High-resolution video created using ProSR. We had to try a lot of different resizing and upsampling factors to get the correct cut.

time_before_after.mp4 - *Left:* Original low-FPS video. *Right:* High-FPS video created by running ProSR on the time-slices we created.

stress_test.avi - Repeated stress-test. Image fed through ProSR 1000 times.

stress_test_GAN.avi Repeated stress-test. Image fed through ProGANSR 1000 times.

5.3. Repository

We couldn't include all the adversarial images and video frames we produced on Gradescope, but they're all available on our GitHub repo. It can be found here: https://github.com/peterhuson/superzoom

It includes our code as well as the input and output images from our testing.