Super Resolution with Convolutional Neural Network

Yihao Lin
tyleryihao.lin@mail.utoronto.ca

Tasbir Rahman
tasbir.rahman@mail.utoronto.ca

Abstract

Super resolution is the process of turning low-definition images into high-definition images by filling in the details. This project utilized convolutional neural networks as opposed to conventional upsampling methods such as bilinear and bicubic interpolation. We implemented as proposed by several similar studies and compared using different interpolations before feeding the input into the network, along with different loss functions and achieved an average PSNR value of 26.947dB and SSIM value of 0.886 on the test set.

Introduction

Super resolution is the process of turning low-definition images into high-definition images by filling in the details. It is a function of very high demand in many fields, such as the medical field and surveillance. Our particular project was aimed at tackling the Single Image Super-Resolution (SISR).

In this project, we set out to implement a X3 super resolution(magnifies with a scale of 3) with a convolutional neural network to improve image quality that performs better than conventional upsampling methods. Conventional upsampling methods can upscale the image in terms of resolution, but they all fail to restore clarity and inferred details. One prominent example is that upscaled images using conventional methods often have pixelated edges. A convolutional neural network sits between interpolation upscaling and SRGANs, which are generative adversarial networks in terms of image quality reproduced and computing power required.

SISR is an extraordinarily ill defined problem. A low resolution image is obtained from a high resolution image by degrading it somehow. such as by reducing the size and then increasing the size with interpolation methods like bilinear, bicubic, and nearest neighbour interpolation methods. Another possibility is by blurring the high resolution image with Gaussian blur. The commonality of these degradation methods is that they result in a loss of detail. The degradation function is a function that takes a high resolution image as input and outputs a low resolution image. What SISR aims to do is find the inverse of this degradation function. What makes this ill defined is the fact that a low resolution image can correspond to multiple different high resolution images.

Literature Review

The first approach to SISR was to use interpolation such as bicubic [1]. This approach has a fast time complexity but the results are usually poor. What results from this approach are jagged edges and overly smoothed textures. Interpolation based methods do not make use of any information beyond what was in the low-resolution image. Thus it is not possible to recover detail from low resolution images using interpolation. As such, this approach is unsuitable for recovering high resolution imagery.

One of the earlier methods involved using sparse coding [2]. A sparse representation of an image is a vector in which very few of its components are non-zero. SISR using sparse coding works like this. A low resolution (LR) image y is a result of this function where x is a high resolution (HR) image .

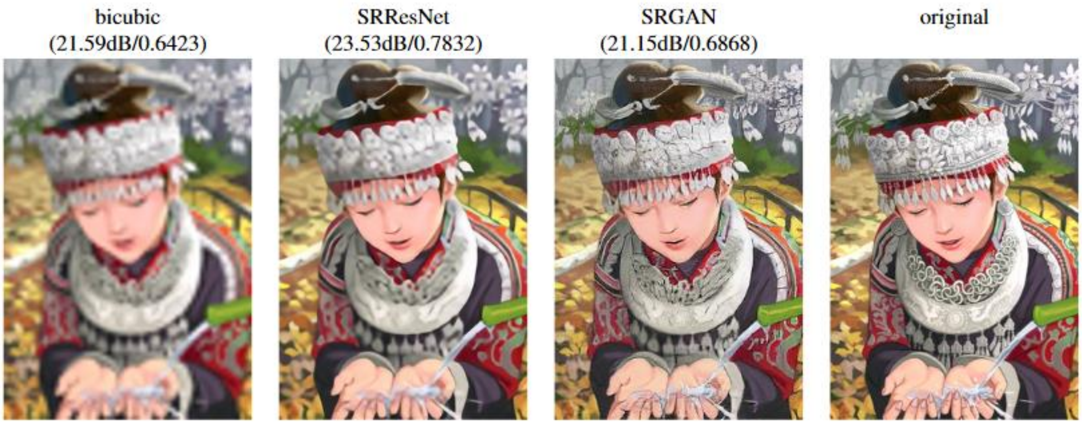
$$y = Lx = LDa$$

Here L is the degradation function, D is a dictionary that converts sparse representations to images, and a is the sparse representation of x. What sparse coding does is try to solve for a. It gets around the problem of having many different HR outputs from a single LR by solving for the *sparsest* possible value of a. The algorithm is a bit more complex in that it uses two dictionaries, one for LR images and another for HR images.

The sparse coding approach has been adapted into a deep convolutional neural network called SRCNN [3]. The layers of the network are based off the steps in the sparse coding approach. This network uses mean squared (MSE) error as its loss function and optimizes using stochastic gradient descent (SGD). Minimizing MSE has been known to favor a high peak signal to noise ratio (PSNR) according to the original SRCNN paper. PSNR is a metric used for estimating the quality of image reconstruction. In the case of SISR this means recreating the HR image. Overall SRCNN has been shown to outperform sparse coding and interpolation based methods.

After SRCNN, another approach has been attempted. Namely, Residual Neural Networks (ResNets) and Generative Adversarial Networks (GANs). ResNets make use of skip connections and have been shown to perform as well shallower networks while not being harder to train [4]. GANs are insprited by game theory. They make use of two networks. The first is the generator which in the context of SISR creates high resolution images and the discriminator which tries to distinguish between images made by the generator and real life images [5]. For SISR they are called SRResnet and SRGAN respectively [6]. These were both defined within the same paper. In this paper it showed that SRResnet was the best of all the networks we mentioned in terms of PSNR.

One key insight made by the SRResnet and SRGAN paper is that PSNR is not necessarily indicative of perceptual quality of an HR reconstruction. They found that optimizing for PSNR (which as previously mentioned occurs with MSE as a loss fun ction) results in an overly smooth reconstruction, which is ineffective for high frequency textures. For SRGAN, they have instead opted for a different loss function instead of MSE on the pixels. Instead they took the feature maps from VGG, a deep neural network image classifier [7] that was pretrained and did MSE on VGG’s outputs. What resulted was images with lower PSNR but better perceptual quality as seen in the image below which was from the paper. Take a look at the white part of the hat and how it’s less smooth on SRGAN than it is on SRResnet and thus closer to the original image.



After SRGAN, a further improvement was made, ESRGAN [8] which had a different structure than SRGAN. It also used a slightly different loss function than SRGAN where VGG features are used before they were subject to activation functions. They found that activated features performed worse due to being more sparse. They also removed batch normalization from SRGAN because they experimentally found that batch normalization introduced artifacts in the reconstructed HR image

Methodology

We implemented SRCNN with some caveats.

We chose to implement SRCNN in spite of better performing alternatives like SRGAN because the structure of the network is far simpler. It also had fewer training parameters, onlny around 8000 meaning it wouldn’t take too long to converge. The paper also used a relatively small dataset, allowing us to save on limited VRAM.

The network was trained on the Y channel of the input patches when converted to the YCbCr color space as was done in the paper. In addition, all intensity values were normalized to be between 0 and 1

Unlike the paper, we used Adam optimization instead of Stochastic Gradient Descent because it is believed to converge faster [9].

With the same network structure and dataset we had three networks trained in different ways.

BCMSE: This is how it was done in the paper. Our loss function was MSE, and our input was a LR image upscaled using bicubic interpolation

NNMSE: We wanted to see the effect of different interpolations. In this case, our loss function remained the same as BCMSE but instead of the input using bicubic interpolation before being fed to the network, we used nearest neighbour interpolation.

BCVGG: We wanted to see if there would be improvement in high frequency cases if we used the VGG perceptual loss in SRGAN as our loss function. This only differs from BCMSE by using the VGG perceptual loss instead of MSE

Each network was trained for 10 hours on an RTX 2060 SUPER video card and a Ryzen 5 3600 processor with 16 GB RAM. This meant we ran BCMSE and NNMSE for 2000 epochs while BCVGG took 500 epochs.

Since VGG used 224x224 RGB images as training inputs, we copied the y-channel three times before inputting it into VGG's feature maps. We also, unlike SRGAN used the feature maps from earlier layers instead of later ones. This is because our label sizes and training outputs were 21x21 and VGG uses maxpooling. We used enough layers so that the feature maps resulting from 21x21 inputs were 2x2 so at least some spatially relevant information would be retained. We believed that the lower number of epochs along with the fact that VGG was trained on higher resolution color images will result in less than ideal results.

For predictions, the Cb and Cr channels are computed with bicubic interpolation. The Y channel is then computed using the neural network. Since the neural network outputs a smaller image, we paste the Y channel in the middle of the bicubic interpolated image. This results in minor border effects for predictions.

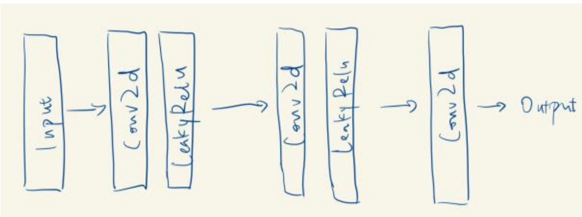
We trained with a scaling factor of 3 like the original SRCNN and also only analyzed results on a 3x scaling factor

Dataset

We selected 91 images from the paper's 91 image dataset. For this implementation, we cropped patches of 33x33pixels with a stride between them of 14 pixels as was done on the paper. This resulted in about 20,000 sub images as our training set. First we downsize the patch by a factor of 3, then upsample them using non-neural interpolation methods (Bicubic like the paper for BCMSE and BCVGG, or Nearest neighbour for NNMSE) to get our training inputs. The result would be the input into the neural network with the original patch being the target. Our labels are center cropped by 21x21 because the network, as it was in the paper, did not make use of any padding at all for it' s convolutional layers, resulting in a smaller output.

We did not use the other datasets that consisted of more images because they achieved only slightly better results in the original SRGAN paper. A larger data set also requires more epochs and more RAM which we were limited in.

Structure



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 25, 25]	5,248
Conv2d-2	[-1, 32, 25, 25]	2,080
Conv2d-3	[-1, 1, 21, 21]	801

Total params: 8,129
Trainable params: 8,129
Non-trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 0.46
Params size (MB): 0.03
Estimated Total Size (MB): 0.50

The convolutional neural network consists of 3 convolutional layers, with LeakyRelu activation after the first two layers. The network has a total of 8129 trainable parameters. The hyperparameters are the same as SRCNN from the paper.

Results

Bicubic BCMSE Ground Truth

Bicubic BCMSE Ground Truth

Bicubic BCMSE Ground Truth

Above are three sets of images for the result showcase. We can see that BCMSE is very good at eliminating pixelated edges, as seen in bicubic interpolations. However, for portrait photos, it still struggles to fill in the details. In addition to this, since we are using interpolated images as the input in the first place, any aliasing that occurred during the initial upsampling will be recognized as part of our image by the BCMSE.

One of the reasons why we chose to do super resolution was that we also wanted to see how pixel arts from games will turn out by passing them into our network. As we can see that the convolution neural network was able to fill in some details and remove most pixelated edges. The results looked surprisingly good. We did not have any corresponding HR images for the pixel art as they do not exist.

Bicubic

BCMSE

NNMSE

BCVGG

Ground Truth

In the tree above, BCMSE overly smoothens the tree’ s texture. Look at the bottom of the tree. It’ s as if the bottom of the tree was water colored on to the image. NNMSE is similar but with more jagged edges. BCVGG, does not perform well either. There does seem to be less of a contrast between the tree and the background, and BCVGG does appear to be noisier. The branches in the tree are lost in all cases, replaced with blobs of green

BCVGG probably would have performed better if we ran it for more iterations, used RGB images instead of Y channel images, used deeper feature maps, and used larger image patches.

Metrics

For training and validation we are using the Mean Square Error:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(h(x^{(i)};\theta) - y(i)\right)^2$$

We also implemented a model using VGG loss which is used in SRGAN implementations. We used a pretrained VGG model trained on 224x224 patches to evaluate as an alternative to pixel-wise losses.

For evaluation of the output images, we are using a metric called the Peak Signal-to-Noise Ratio (PSNR) which is the ratio between maximum power of a signal to power of corrupting noise, expressed in logarithmic decibel scale. Its equation is:

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

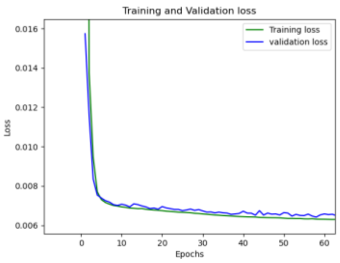
The minimization of MSE means the maximization of PSNR value. One thing to note here is that MAX stands for the maximum possible power of a pixel which should be 255. However, in this implementation we normalize the pixel values to (0,1). Hence, we used 1 as MAX here.

PSNR is heavily pixel-to-pixel based and concentrates on pixel colours, which means that any little change to brightness will reduce PSNR by a substantial amount. Therefore, we decided to use a second metric called Structural Similarity Index Measure(SSIM) which focuses on image structure.

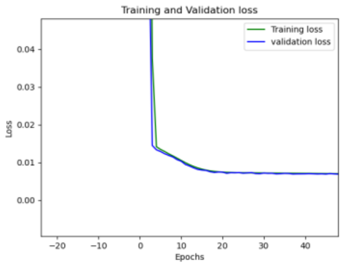
$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

In this implementation we are using the Skimage Python package for SSIM metrics.

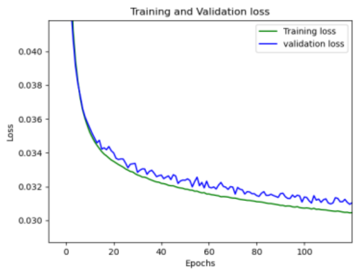
Analysis



NNMSE



BCMSE



BCVGG

NNMSE is the network trained with MSE and nearest neighbor interpolated patch as input. BCMSE is the network trained with MSE and bicubic interpolated patch as input. BCVGG is the network trained with VGG loss and bicubic interpolation patch as input.

The training and validation losses converged very quickly, the reason for this is that we are using Adam’s gradient descent for calculating the learning rate instead of the usual stochastic gradient descent.

It seemed that BCMMSE was the fastest to converge and was also the most stable for validation error. NNMSE also converged but the convergence was less stable for validation error suggesting a worse performance when using nearest neighbor instead of bicubic interpolation

Unfortunately it seemed that we stopped BCVGG before it could really converge. We only ran BCVGG’s training for 500 epochs since that many epochs took around the same amount of time to complete as 2000 epochs for NNMSE and BCMSE. Our machine did not have good cooling so we did not want to run the network for more than 10 hours at a time.

Below is the result of our neural networks after training for 500 epochs for VGG and 2000 for other nets. The test set is a selection of 10 random images.

	BC	NNMSE	BCMSE	BCVGG
Baby	31.82dB	27.79dB	32dB	30.61dB
	0.94	0.89	0.94	0.92
Bird	30.56dB	25.69dB	30dB	27.3dB
	0.95	0.89	0.95	0.88
Butterfly	20.87dB	18.34dB	22.08dB	21.1dB
	0.85	0.80	0.89	0.86
Dinesh	27.82dB	24.19dB	28.89dB	27.6dB

	0.94	0.89	0.95	0.93
Head	29.98dB	26.41dB	30.13dB	26dB
	0.86	0.71	0.83	0.93
Kaladin	29.11dB	25.86dB	27.03dB	27.02dB
	0.93	0.89	0.93	0.92
Ninja	31.28dB	28.71dB	27.51dB	23.66dB
	0.95	0.92	0.90	0.80
Tree	20.40dB	19.71dB	20.83dB	19.7dB
	0.72	0.66	0.72	0.69
Wolf	25.01dB	24.09dB	25.63dB	24.93dB
	0.80	0.77	0.81	0.79
Woman	25.72dB	21.26dB	25.37dB	24.3dB
	0.92	0.84	0.93	0.86
AVERAGE	27.257dB	24.205dB	26.947dB	25.222dB
	0.886	0.826	0.886	0.858

We are only comparing with bicubic interpolation here since it produces the best results in non-neural options. We can see that NNMSE produces worse results than bicubic interpolation. The main reason is that we used NN interpolated patches as our input which put the network at a disadvantage in the first place. Theoretically it is better than regular nearest neighbor interpolation but we mainly trained this network for comparison with our other ones.

BCMSE seems to be the best performing network out of the three. It is able to achieve similar average PSNR and SSIM as bicubic interpolation. It performs way better than bicubic on specific images such as the butterfly. A possible reason for this may be that our training dataset contains a lot of images with patterns(insects, foliage, etc.) The prediction looks better with the removal of pixelated edges.

Bicubic BCMSE Ground Truth

Another thing we observed is that if aliasing has happened in the upsampling using interpolation, our network would see that as part of the image feature and tries to add details to that. The result is that our network can potentially produce worse predictions. One prominent example is the ninja image where aliasing has happened in the non-uniform region on the hood. Our network tries to restore the definition of that region but is not able to.

Bicubic BCMSE Ground Truth

As for our model trained with VGG loss, it can potentially produce better results if we train it for more epochs, but the results it produces now is still noisy.

Conclusion

In our implementation, we demonstrated that super resolution with convolutional neural networks is a good alternative to non-neural options like interpolations. The networks are able to add detail properly and produce better looking predictions. However, issues still exist. Our implementation is limited by its training input. We can still see remaining pixelation, such as in the pixel art example. It also struggles with complex detail, such as non-uniform surfaces and facial details. This mainly comes down to the fact that we are removing much information by downsample and upsampling the images for our training input.

We expect that with a larger dataset and more computing power for more epochs, the performance of the networks can be improved upon. Since we are training on a RTX gaming GPU, we only used a shallow network of 3 convolutional layers. Therefore, changes to the structure of the network may also increase prediction performance.

Alternatively, other better performing implementations of super resolution exist, namely SR with residual networks or generative adversarial networks. However, they require much more computing resources. In the future, we would like to approach super resolution with these two implementations, possibly with the help of AWS.

Contributions

This implementation is done by our team of two. Lin focused on preprocessing the images for the network for both predictions, and training, along with making predictions and measuring metrics for the test set. Tasbir focused on making the actual network itself, training the network, and incorporating VGG loss.

References

[1] Hsieh Hou and H. Andrews, "Cubic splines for image interpolation and digital filtering," IEEE Trans. Acoust., Speech, Signal Process., vol. 26, no. 6, pp. 508–517, Dec. 1978, doi: 10.1109/tassp.1978.1163154. [Online]. Available: http://dx.doi.org/10.1109/TASSP.1978.1163154

[2]Jianchao Yang, J. Wright, T. Huang, and Yi Ma, "Image super-resolution as sparse representation of raw image patches," in 2008 IEEE Conference on Computer Vision and Pattern Recognition, 2008, doi: 10.1109/cvpr.2008.4587647 [Online]. Available: http://dx.doi.org/10.1109/CVPR.2008.4587647

[3] C. Dong, C. C. Loy, K. He, and X. Tang, "Image Super-Resolution Using Deep Convolutional Networks," IEEE Trans. Pattern Anal. Mach. Intell., vol. 38, no. 2, pp. 295–307, Feb. 2016, doi: 10.1109/tpami.2015.2439281. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2015.2439281

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, doi: 10.1109/cvpr.2016.90 [Online]. Available: http://dx.doi.org/10.1109/CVPR.2016.90

[5] I. G. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," *Advances in neural information processing systems.* , pp. 2672–2680. Available: https://papers.nips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[6] C. Ledig et al., "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," in 2017 IEEE Conference on Computer Vision and Patter

n Recognition (CVPR), 2017, doi: 10.1109/cvpr.2017.19 [Online]. Available: http://dx.doi.org/10.1109/CVPR.2017.19

[7] Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv:1409.1556 [cs], Sep. 2014

[8] X. Wang et al., "ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks," in Lecture Notes in Computer Science, Springer International Publishing, 2019, pp. 63–79 [Online]. Available: http://dx.doi.org/10.1007/978-3-030-11021-5_5

[9] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv:1412.6980 [cs], Dec. 2014