A Comparative Analysis of multiple works on Single Image Super-Resolution

Selected Topics in Computer Science

Aman Shenoy
2016A8PS0393P
f2016393@pilani.bits-pilani.ac.in

Prajjwal Mahajan
2016A7PS0123P
f2016123@pilani.bits-pilani.ac.in

1 Problem Statement

[1] Single Image Super Resolution is one of the oldest problems in computer vision. It aims to be able to get a high resolution image from any given low resolution image, with minimal loss in information. This problem is inherently ill-posed (*No unique solution to any given problem*) since multiple valid solutions can exist for any given low resolution image.

The advent of deep learning, particularly that of learned convolutional neural networks [4], has shined new light on this problem due to the immense power of deep learning algorithms, especially when given abundant training data. Since one can easily take any high resolution image and convert it to a low resolution image, any set of high-resolution images can act as a training set, making this problem inherently very well suited for deep learning based methods.

Further, with the recent introduction of Generative Adversarial Networks [2], Single Image Super Resolution has further leaped significantly due to the incredibly high perceptual quality of images that can be generated by a GAN.

In this report we provide a comprehensive analysis of two standard methods for Single Image Super Resolution − namely SRCNN [1] and SRGAN [5]. We provide an in depth analysis of the methods additional to the results already given in the respective papers, and provide a thorough comparitive analysis between multiple methods for Single Image Super Resolution.

1.1 Implementation from Scratch

Since these methods have been implemented in the *Caffe* framework and also do not provide interactive visualisations, we have implemented SRCNN from scratch with visualizations of intermediate layers in the form of **.gif** files and also provide the training and testing trends for multiple metrics.

The re-implementation of this paper was written in the *PyTorch* framework and our github repository can be found at

`https://github.com/amanshenoy/image-super-resolution/tree/stcs`

The above repository contains a completely re-implemented training script, inference script, results, pre-trained models, and all demonstrations[2].

[1] For this report we have followed the NeurIPS paper style available publicly on their website

[2] All demonstrations and visualizations were created using the open-source project **ffmpeg**

2 Survey on the Topic

Before Deep Learning methods were commonplace, super-resolution algorithms would commonly involve finding low-res $\rightarrow$ high-res mappings through multiple patch-based methods such as Nearest Neighbours, Kernel Regression, Random Forests, or Anchored Neighbourhood Regression. Patch-based methods used a small patch of the image for training instead of the whole image and then reconstruct the image by using the function that has been learned on patches. This is not only able to give us a large number of samples for training, but also reduces system memory requirements.

Other methods included dictionary based methods, where a patch was transformed to a higher dimensional space and then mapped again to a lower dimensional space, in hopes of achieving passable super resolution.

Even though machine learning techniques and algorithms have long been used to be able to learn super-resolution functions, the use of deep learning based learning achieved remarkable comparative performance. Amongst these, SRCNN [1] was one of the first to formulate the problem in such a way that the super-resolution function could be written purely in terms of learned convolution operations. This gave a clean formulation of the problem with each convolution operation holding interpret able meaning.

These early convolution based methods learned parameters through stochastic gradient descent on a loss function of **Mean Squared Error**. Minimizing the MSE between the high-res image and the output of the model maximized a key metric for super-resolution $-$ the **Peak Signal to Noise Ratio (PSNR)**. The PSNR of an image, given the MSE between the normalized $(0 \rightarrow 1)$ images, is

$$PSNR = 10 \times log_{10}(1/MSE) \tag{1}$$

PSNR and SSIM (Structural Similarity) were considered to be two of the most important metrics for super resolution. Since minimizing MSE directly increases PSNR, it was the most common loss function used by almost all super resolution algorithms, till recently.

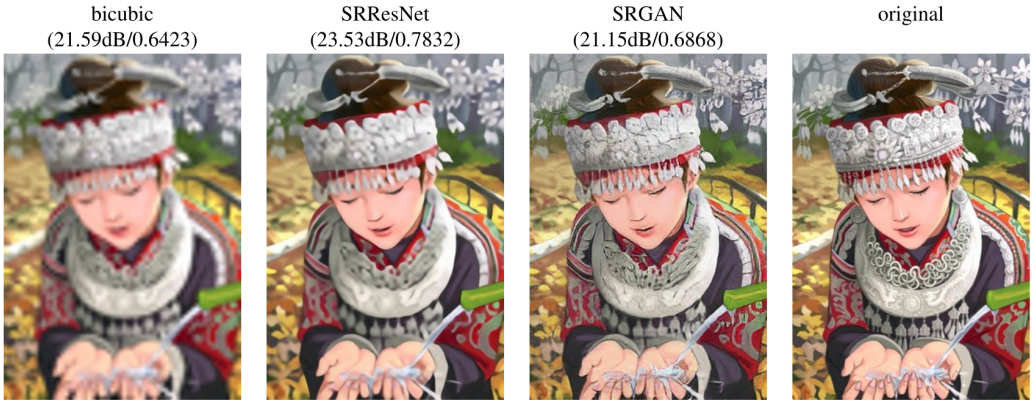

Figure 1: Comparative Performance of multiple Super Resolution Algorithms with their PSNR values

More recently it was observed that PSNR as a metric can potentially be deceptive and need not be representative of perceptual quality. This can be seen above where SRResNet [5] has a better PSNR than SRGAN [5], but is not perceptually better. PSNR scores well when the MSE between the images is low, which need not be a good indicator of accurate reconstruction. Training with MSE learns to average over multiple values in the image-space manifold containing all the accurate reconstructions. This causes the reconstruction learned using MSE to be very smoothed and hence fail to capture parts of the image that have a very high variance.

To be able to deal with this issue, SRGAN [5] uses a perceptual loss which consists of two components $-$ content loss and adversarial loss. The content loss makes sure that the content in the reconstructed image is the same (MSE can be used here) and adversarial loss makes sure that the generated high-res images have high perceptual quality.

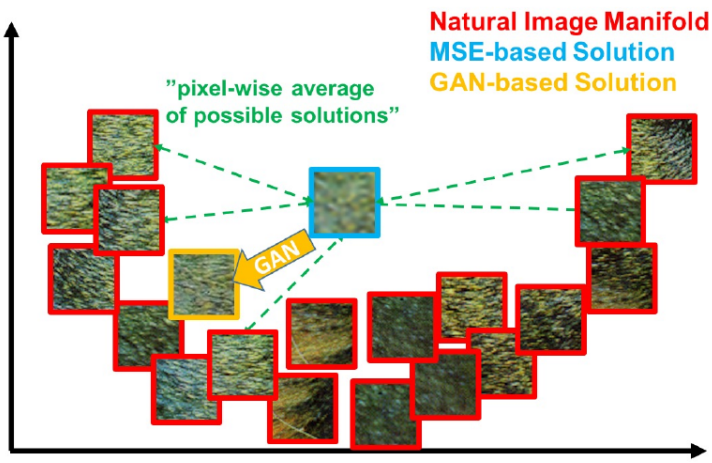

Figure 2: Illustration of patches obtained by MSE and GAN's and how they relate to the natural high-res image manifold

The reformulation of the loss function in SRGAN [5] proved to be crucial, as perceptual quality in super resolution increased drastically. Since the adversarial loss is minimized purely on the basis of its ability to trick a discriminator, this becomes a very good judge of image quality. This pushes the image to have higher variance and be sharper, giving overall better quality results for super-resolution.

3 Architectural Details

Apart from vastly different architectures, both of the models being studied in this report also have vastly different ways of training. The following section will introduce both architectures and also explain the loss formulation and training.

3.1 Pre-Processing

The only pre-processing that was done was generation of the dataset being used. Patches were extracted for every image. In our code (`https://github.com/amanshenoy/image-super-resolution/tree/stcs`), we have used patch size $ps = 33$ as done by the paper to be able to compare results. For every image, 5 patches were extracted by our code $\rightarrow$ 4 corner patches and 1 center patch.

Once patches are taken, our dataset should now have $5 \times originaldatasetsize$ number of high-res patches. These patches are then down-scaled by 2 using bicubic interpolation and then up-scaled again by bicubic interpolation to give us a low-res counterpart of that patch, giving us a dataset of low-res, high-res pairs to be able to train. The interpolation methods can be changed and bicubic interpolation was used only because the paper did so. All this pre-processing and augmentation techniques can be found in our training notebook (`https://github.com/amanshenoy/image-super-resolution/blob/stcs/training.ipynb`).

Both models would roughly follow the same pre-processing, with no additional supervision of any sort required for both models. The only difference between the models is the way the supervised data is used, and both work on the exact same kind of data.

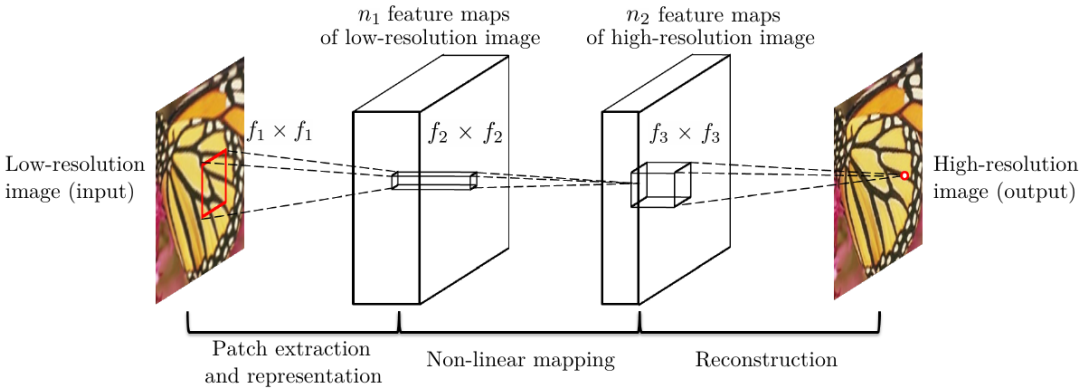

Figure 3: SRCNN model architecture

3.2 SRCNN

3.2.1 Architecture

SRCNN [1] was one of the first models to model the super resolution problem using learned convolutions. This model was introduced around the time where patch based dictionary mappings were dominantly used for image super-resolution. This model had used dictionary mappings as was commonly used at the time, but the authors were able to rewrite the function purely in terms of convolutions and piece-wise linear functions.

SRCNN formulation was initially done from a dictionary mapping perspective. They had followed a 3 stage process in their original formulation.

1. **Patch extraction and representation**: this operation extracts (overlapping) patches from the low-resolution image Y and represents each patch as a high-dimensional vector. These vectors comprise a set of feature maps, of which the number equals to the dimensionality of the vectors.
2. **Non-linear mapping**: this operation non-linearly maps each high-dimensional vector onto another high-dimensional vector. Each mapped vector is conceptually the representation of a high-resolution patch. These vectors comprise another set of feature maps.
3. **Reconstruction**: this operation aggregates the above high-resolution patch-wise representations to generate the final high-resolution image. This image is expected to be similar to the ground truth patch.

The paper further proceeded to show that this formulation of the problem can appropriately be implemented as convolutions. Each stage can be shown to be equivalent to a convolution followed by a piece-wise linear function. This would reformulate the problem as

low-res patch $\rightarrow Conv(f_1, n_1) \rightarrow ReLU \rightarrow Conv(f_2, n_2) \rightarrow ReLU \rightarrow Conv(ps, 3) \rightarrow reconstruction$

Where $Conv(a, b)$ is a convolution operation with filter size a and b channels; and ps is patch size.

3.2.2 Training

The original paper was trained using MSE as a loss function using stochastic gradient descent to optimize the kernels. In our codes training we have instead used Adam [3], for earlier and better convergence. The comparison is later spoken about in further sections.

We have also used mini-batch training instead of backward steps on single examples. This reduces training time and also betters generalization.

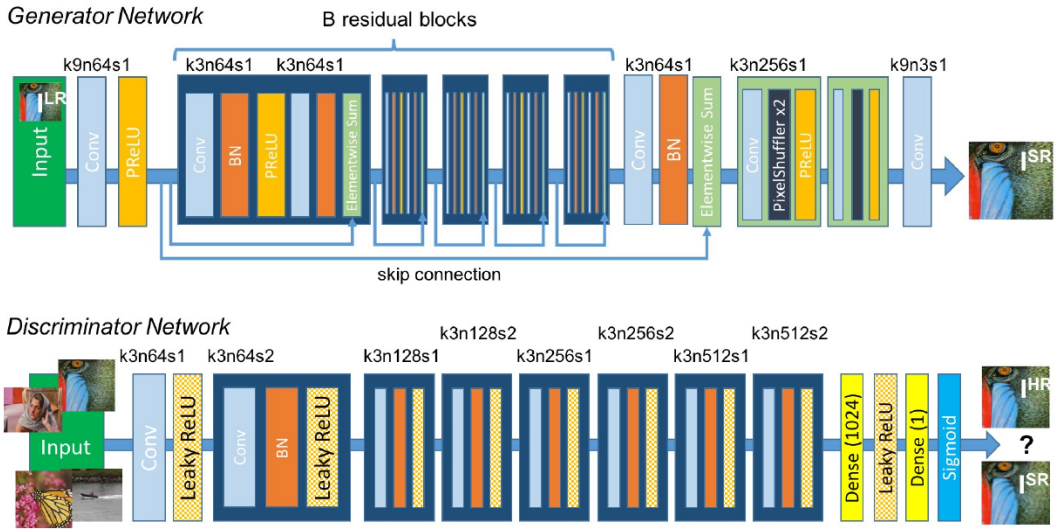

Figure 4: SRGAN discriminator and generator architecture

3.3 SRGAN

3.3.1 Architecture

There are multiple architectures that were introduced in the paper that introduced SRGAN [5]. We will focus only on the best model mentioned in the paper, which is SRGAN [5] itself.

SRGAN [5] followed a GAN [2] based objective, where it was trained on a perceptual loss instead of directly on MSE. This perceptual loss had two components $\rightarrow$ adversarial loss and content loss. Since the model is GAN [2] based, we need to establish architectures for both the discriminator and the generator. The generator was used as the super resolving function and both the architectures are demonstrated above.

The Generator was a ResNet based architecture with skip connections and the Discriminator was taken from some of most popular work on GAN's [2], following reccomendations to not use pooling and to use LeakyReLU with $\alpha = 0.2$

3.3.2 Training

The effectiveness of SRGAN [5] comes primarily from the way it was trained. It was observed that minimizing MSE need not learn visually pleasing results as explained before. A perceptual loss was proposed to be able to learn accurate and perceptually appealing results.

The perceptual loss had two components - an adversarial loss and a content loss. This perceptual loss is described in equation form below.

$$L_{perceptual} = L_{content} + \alpha L_{adversarial} \tag{2}$$

Where α was a hyper-parameter whose optimal value was noted to be 10^{-3}.

Content Loss is what assures that the content between the image and its reconstruction is maintained to be very similar. The older SR models had $\alpha = 0$ and was trained on only a content loss. The most obvious content loss would be MSE, as it does exactly whats needed to maintain similarity in content. Even though this instinct is fairly justified MSE seems to be a bad option for this. This is because MSE only assures pixel wise similiarity and completely ignores the content of the patch.

The alternative suggested by the paper was to use a **VGG loss**, where both images were inferred on with a pre-trained VGG-19 model, and the penultimate layer representation was considered a content

representation of the image. The euclidean distance between this representation for the high-res patch and reconstructed patch was taken to be the content loss.

This formulation assures that the images stay similar by virtue of their content and not by virtue of their pixel-wise similarity, which is crucial in being able to get sharp images.

Adversarial Loss ensures that the image quality is very high by pushing the parameters of the super resolution function to take values that predict images in a high resolution manifold. The adversarial loss only tries to make the image quality high and is not concerned if it matches with the input image.

Adversarial training for the generator is performed as with inspiration from DCGAN [6] architecture. The discriminator is first trained for k backward passes using a sigmoid loss and then the generator is trained on $L_{perceptual}$. This training happens in an alternating fashion.

4 Comparative Study

On comparing both the approaches (SRCNN [1] and SRGAN [5]) one can very easily see that SRGAN [5] addresses a lot of the downfalls of SRCNN [1]. SRGAN understood that PSNR need not be reflective of the perceptual quality of images and gave examples where PSNR can be deceptive.

On comparing the ideas that went into both the models, one can safely say that SRCNN being older takes a lot of ideas from classical computer vision, whereas SRGAN uses modern deep learning approaches. The explanation of certain aspects of SRCNN would involve knowledge of patch based dictionary mappings and the deep learning approach was only a convenient reformulation. SRGAN on the other hand used some very standard ideas in Deep Learning such as a loss being a linear combination of multiple loss functions, using an intermediate representation from a pre-trained model, adversarial training for perceptual quality etc.

Since both models are trained on patches of $ps = 33$, the **time complexity** of inference is the number of patches that can fit in that image for both models. If an image has shape $W, H, 3$, then the time complexity would be $O(WH)$ since number of patches in an image $\sim WH$. Both algorithms will vary in the constant terms in the complexity due to the different depth in the models.

Both models are extremely **memory efficient**. SRCNN [1] being extremely small, has very few parameters and SRGAN [5] uses a lot of pre-trained models, avoiding the need to store gradients for those parameters.

Eval. Mat	Scale	Bicubic	SC [50]	NE+LLE [4]	KK [25]	ANR [41]	A+ [41]	SRCNN
PSNR	2	33.66	-	35.77	36.20	35.83	36.54	**36.66**
	3	30.39	31.42	31.84	32.28	31.92	32.59	**32.75**
	4	28.42	-	29.61	30.03	29.69	30.28	**30.49**

Figure 5: SRCNN test performance on the Set5 dataset compared to non deep learning approaches

	SRResNet-		SRGAN-		
Set5	MSE	VGG22	MSE	VGG22	VGG54
PSNR	32.05	30.51	30.64	29.84	29.40
SSIM	0.9019	0.8803	0.8701	0.8468	0.8472
MOS	3.37	3.46	3.77	3.78	3.58

Figure 6: SRGAN variants test performance on the Set5 dataset with scale 4

As one can see from the tables, SRGAN did not improve PSNR too much and actually performs better with the content loss being MSE instead of VGG54. To account for the fact that these metrics can be deceptive in nature, SRGAN paper introduced a new metric called MOS (Mean Opinion Score). Since SRCNN does not record MOS, we have not put up the MOS scores.

Figure 7 is a demonstration of how the PSNR and SSIM scores can potentially be deceptive.

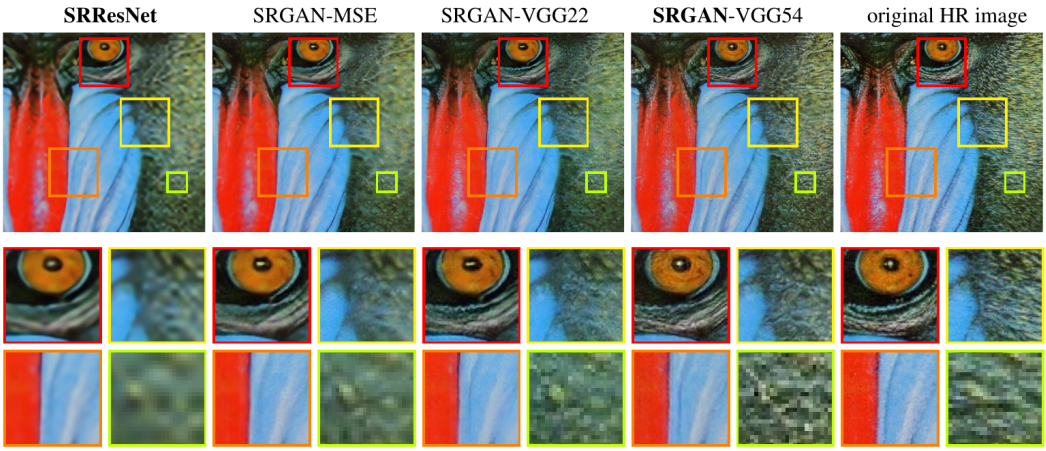

Figure 7: Demonstration of results, showing how PSNR's very similar can look very different perceptually

5 Improvements

Due to the large nature of SRGAN, our code base only worked on SRCNN. We rewrote the model from scratch to make improvements. Our version of SRCNN was trained in Google Colab and with minor tweaks was able to better performance as compared to the original paper. The code-base consisting of all results, demonstrations, training, and inference can be found in the below link.

`https://github.com/amanshenoy/image-super-resolution/tree/stcs`

Some changes as compared to the paper were

1. Adam optimizer [3] was used instead of Stochastic Gradient and training was done in batches. This allowed better and earlier convergence.
2. Skip-connection was added from input to output for the model to be able to start at an appropriate initialization, learning only the difference image instead of the whole image. This sped up training and also bettered the results.
3. Intermediate channels were consistently visualized and videos were made for how channels change over time and how the model trains over time.
4. Increased the number of channels in the intermediate layers for the model to be able to learn more features.

These improvements not only sped up training but also beat the results in the paper. Visualizations were made to be able to interpret what the model is doing. All visualizations were made using ffmpeg (`https://ffmpeg.org/`) and were generated by ourselves on dataset STL-10 (`https://ai.stanford.edu/~acoates/stl10/`).

A collective list of all visualizations we generated can be found at `https://github.com/amanshenoy/image-super-resolution/tree/stcs/demonstrations` and all generated results can be found at `https://github.com/amanshenoy/image-super-resolution/tree/stcs/results`[3]

6 Results

The results of the papers, apart from being shown above, are comprehensively described in [1] and [5]. We use this section to demonstrate our results. We show that with the improvements made by us mentioned in the previous section, we perform better than the original paper.

[3] GitHub links have been given for convenience even though the code base has been submitted with the report.

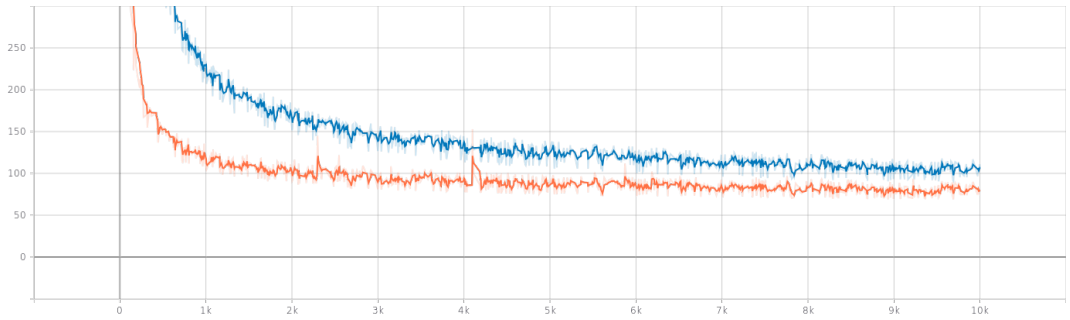

Figure 8: Our Mean Squared Error for training (Orange) and testing (Blue)

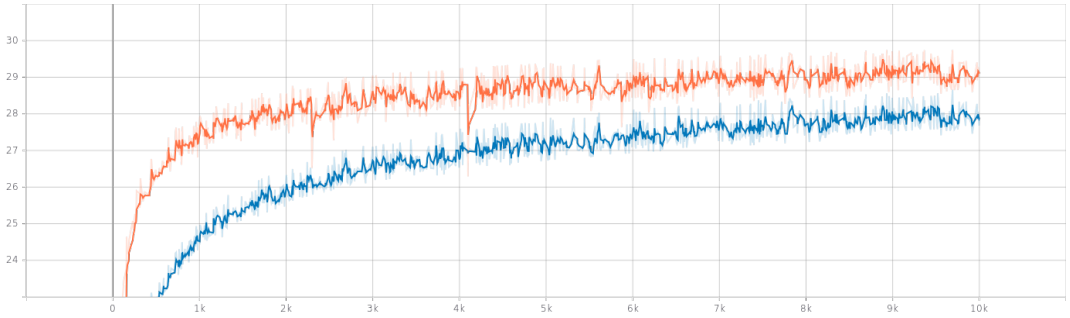

Figure 9: Our PSNR for training (Orange) and testing (Blue)

As visible in figure 9 our testing PSNR comes to 28.8dB, which is higher than the 27.95dB achieved by the paper on the same network. Our model also outperforms the paper on standard individual images. Results on all these standard images can be found at this link (`https://github.com/amanshenoy/image-super-resolution/tree/stcs/results`) and in the folders given with this report.

We also plotted the average PSNR for bicubic interpolation to be able to compare with our results. All the plots were generated using tensorboard (`https://www.tensorflow.org/tensorboard`) and code to make the graphs is in the training notebook (`https://github.com/amanshenoy/image-super-resolution/blob/stcs/training.ipynb`).

The code can be run on any image of your choice, either locally saved image or through an image link. Complete instructions on how to run the code and visualizations can be found in the readme file with the code-base (`https://github.com/amanshenoy/image-super-resolution/blob/stcs/readme.md`)

When the code is run on an image of choice a reconstruction loop is called, which goes to all the 33×33 patches of the image and run the super resolution function on it and uses the outputs to recon-

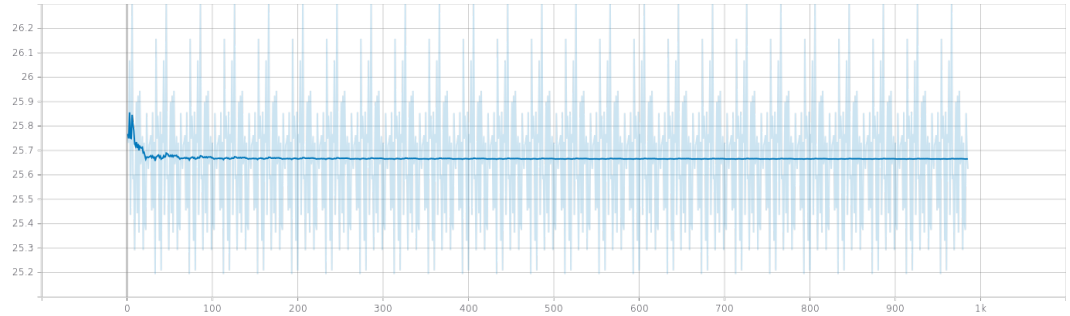

Figure 10: PSNR of bicubic interpolation for comparison

struct. Visualization of this top to bottom progressive scan can be found at `https://github.com/amanshenoy/image-super-resolution/blob/stcs/demonstrations/progressive.gif`.

7 Conclusion

On a concluding note, this report was a comparative analysis of two very standard approaches to Single Image Super Resolution. We described both these techniques in detail and did a comparative analysis of the two approaches.

Along with this we also made improvements to SRCNN [1] and beat their results on STL-10 dataset. We did a thorough analysis of the network, with visualizations and demonstrations (cannot put .gifs in .pdf) which has been submitted alongside this report. Please do check the code repository out (GitHub recommended since the readme includes some visualizations)

`https://github.com/amanshenoy/image-super-resolution/tree/stcs`

The repository also contains instructions on running the code on any image (link or local).

References

[1] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38, 12 2014.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 06 2014.

[3] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[4] Yann LeCun. Deep learning convolutional networks. pages 1–95, 08 2015.

[5] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. 09 2016.

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.