Super Resolution
Project Report

Florian Jörg
Viviane Tanner
Adrian Wälchli

May 30, 2017

1 Super Resolution

Super resolution is a term used to describe methods for image upscaling. Those methods generate a high resolution output from a low resolution input image. To achieve this, missing data has to be interpolated.

It's possible for two different but very similar high resolution images to have the same low resolution image. Therefore super resolution is an ill-posed problem that does not have a unique solution.

Neural networks present an easy way to handle such problems. They learn a probability distribution over inputs and outputs. Because of that they can be reasonably trained even when there are ambiguities.

Furthermore the Super Resolution Neural Network (SRNN) does not have to work on all inputs (e.g. there is no use in upscaling an image that contains only noise). The domain it works on can be significantly reduced, increasing its accuracy. For example a SRNN used in neuroscience only has to be able to upscale pictures of nervous tissue, knowledge about how to upscale faces would be useless to it.

The first to use neural networks for super resolution were Dong et al. [2]. They only used three convolutional layers. The first extracts low resolution feature maps, the second maps those to high resolution feature maps, and the last reconstructs the high resolution image. Even this simple network was already able to outperform other state of the art algorithms of that time.

Since then many different SRNNs were proposed. Most fit in one of two categories: They either directly upscale the image (like Dong et al. [2]) or they use a naive upscale algorithm (e.g. bicubic interpolation) and then try to improve the quality of the intermediate output with a neural network.

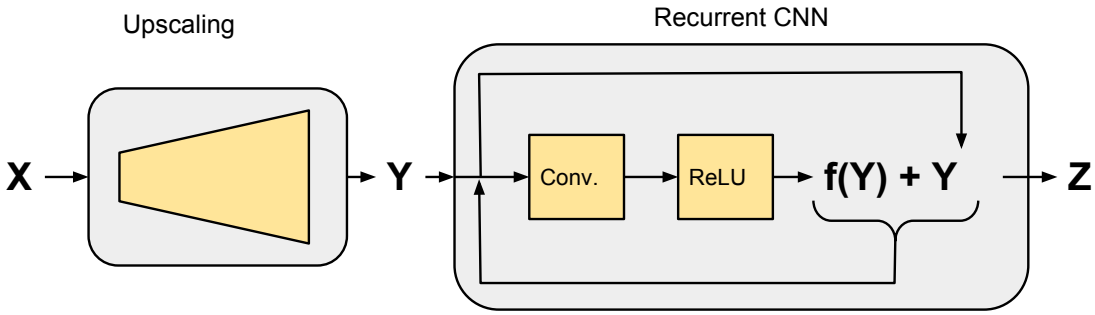


Figure 1: Our main network

2 Proposed Network Architecture

As can be seen in Figure 1 and Table 1, our proposed architecture consists of two parts:

- Upscaling: Convolutional layers that perform upscaling to the desired image size.
- Refinement: A recurrent neural network that adds more detail to the upscaled image in an iterative manner using residual learning.

With this architecture we attempt to combine both approaches to SRNN currently in use.

Part	**Name**	**Type**	**Kernel Size**	**Out Channels**
Upscale	conv1	Convolution	3	6
	conv2	Convolution	3	6
	conv3	Convolution	5	32
	conv4	Convolution	3	$3 * upscale^2$
	shuffle	PixelShuffle	-	3
Recurrent	conv5	Convolution	3	6
	conv6	Convolution	3	3

Table 1: These are the layers used in our network. There is an additional ReLU layer after each convolutional layer except for conv4. The layers listed in the recurrent part form the inside of the recursion.

2.1 Variations

We conduct two ablation studies to investigate the impact of our design choices in the network architecture. Additionally, we train the full network in two separate steps to understand which part trains and converges faster. Finally we tried to make the two parts of the network deeper.

Upscale Only In order to find out how effective our refinement network is, we simply remove it and compute the metrics for comparison. This version of the network is very similar to the one proposed in Dong et al. [2].

Omitting the Residual Connection The residual part of the network aims to add details to the image that could not be learned by the first part. By removing it, we expect the network to perform worse in regions where the network should reconstruct texture detail.

Two-Step Training When training the full network at once, it could be that the recurrent block has a bad influence on the training of the upscale part. In order to make sure that this is not the case, we propose to train in two stages. In the first step, we only train the upscale part as described for the upscale only variation. Then, the second part is trained with the weights of the first part fixed. Finally, we compare the metrics with the other variants.

Deeper Network With a deeper network, we add complexity to our model and expect the network to learn more variations of detail in the data. We added two additional convolution layers in the upscale network and one in the refinement part. The exact layout of the deep network is presented in table 2. In this iteration, we also enlarged the kernel sizes of the first layers.

3 Datasets

The datasets we used were Set5 [1], Set14 [6], BSD100 [5] and Urban100 [3].
Set5 and Set14 are small datasets (5 and 14 images respectively) without any particular theme to them. We used those for small tests and proofs of concept.
BSD100 and Urban100 are larger datasets of 100 images each. BSD100 contains mostly nature based outdoor images, often containing humans or animals. Urban100 consists of architectural images, mostly of modern buildings.

Part	Name	Type	Kernel Size	Out Channels
Upscale	conv1	Convolution	11	6
	conv2	Convolution	7	6
	conv3	Convolution	3	12
	conv4	Convolution	5	16
	conv5	Convolution	3	32
	conv6	Convolution	3	$3 * upscale^2$
	shuffle	PixelShuffle	-	3
Recurrent	conv7	Convolution	9	8
	conv8	Convolution	5	8
	conv9	Convolution	5	3

Table 2: These are the layers used in the deeper version of our network. There is an additional ReLU layer after each convolutional layer except for conv6. The layers listed in the recurrent part form the inside of the recursion.

4 Implementation

To accommodate all those different network and training variations we decided to build a modular framework. The core of the framework is the main program. It loads parameters from the command line and from a parameter file. According to the parameter it then loads the required code modules and executes them. There are five basic modules:

- Network creation: Creates a new network based on a network file declared in the parameters and saves it to disk
- Load training data: Loads training and validation data and prepares it for use
- Train network: Loads a network from disk, trains it, and saves it back to disk periodically to avoid data loss
- Load test data: Loads test data and prepares it for use
- Test network: Loads a network from disk and measures its performance on given test data sets

This modular structure makes it very easy to adapt to different experiments. For example, to implement the variant where the two network parts have to be trained separately only requires writing a new training module. The framework is written in *lua/torch*. For the

recurrent part we used the RNN library by Element-Research.[1] We use Git for version control.[2]

5 Challenges

During training we faced a few pitfalls:

- Vanishing gradients: Even though our residual part is only five layers deep we observed gradients shrinking unnecessarily. To reduce the influence of this on the overall training we separated the network in its two parts and trained them separately. Thus the first part could be trained properly. This produced better inputs for the second part and improved the results.
- Border artefacts: Because of the standard zero padding used in the torch convolution layers our output images suffered from ringing artefacts along the borders. Simply putting a repetition padding layer before every convolutional layer instead of using the built in zero padding resolved this issue.
- Resource usage: The residual part of the network works on full sized images. Because of that the activation vectors of these layers were very large. This not only used up much of our RAM, it also slowed down the training significantly. Doubling the amount of channels in these layers increased training time by a factor of five to ten. Because of that we could not use more channels or layers in the recurrence than we did.
- Non convergence: Some random initialisations of networks were so unlucky, that the training did not converge. A new initialisation of the network always solved this problem. But the time used to train the unlucky network was gone anyway.

[1] RNN library by Element-Research: `https://github.com/Element-Research/rnn`
[2] Source code and results: `https://github.com/awaelchli/ML-FS2017-group-project`

6 Results

Network Variation	Main Net Zero-Pad	Main Net Rep-Pad	No Residual	Upscale Only	Two-step Training	Deep Net
Loss	0.003644	0.006659	0.083311	0.003103	0.003103	0.003740
PSNR	25.4699	22.2475	10.8441	26.1877	26.1892	25.3926
SSIM	0.793297	0.791507	0.392628	0.811401	0.811404	0.795097

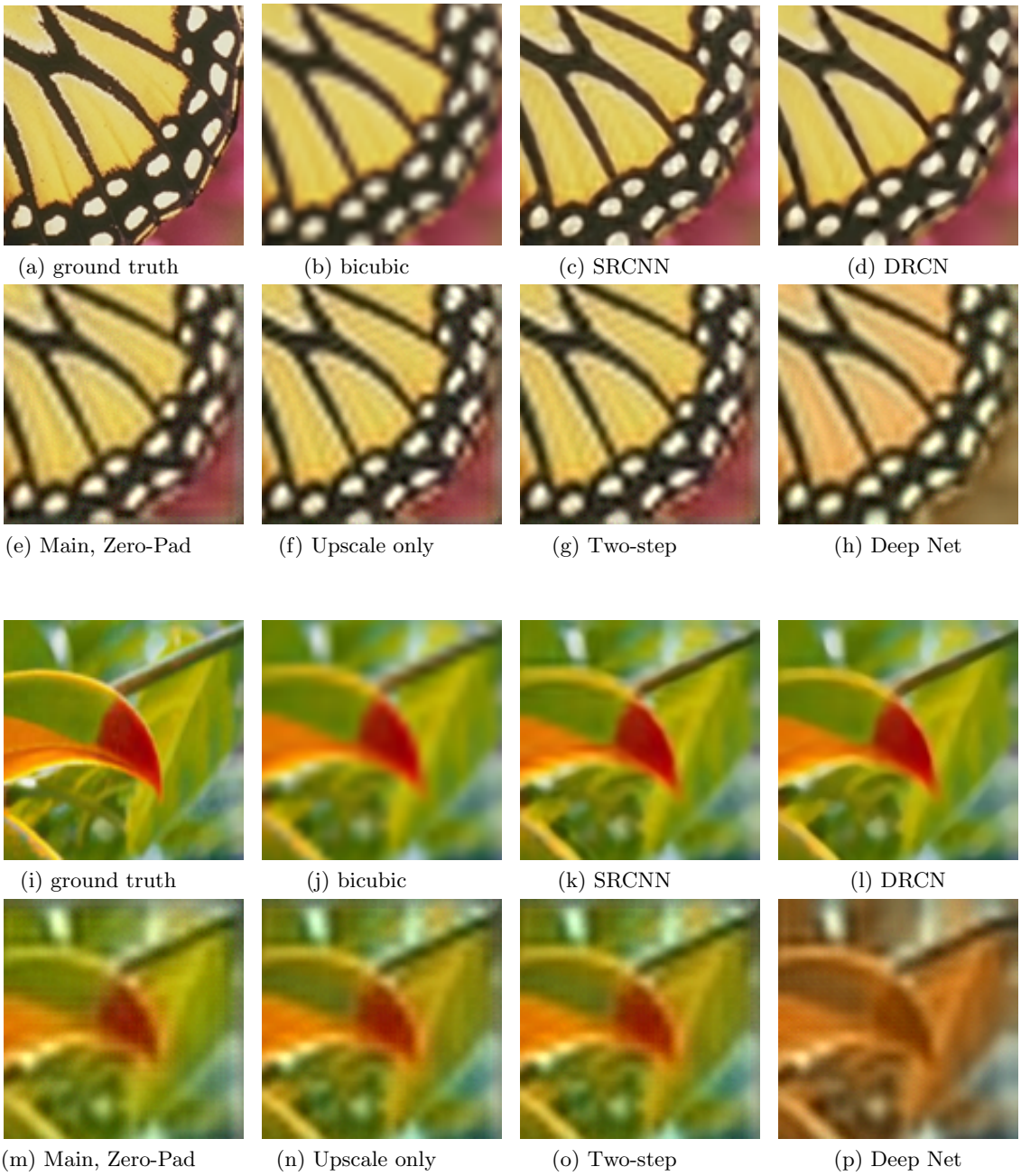
Table 3: Comparison of different network architectures on Set5.

Network Variation	Main Net Zero-Pad	Upscale Only	Two-step Training	Deep Net
Loss	0.008813	0.008290	0.008291	0.009013
PSNR	21.5401	21.8629	21.8624	21.2633
SSIM	0.641027	0.660270	0.660288	0.654327

Table 4: Comparison of different network architectures on Urban100.

Data set	Bicubic (ours) PSNR/SSIM	Bicubic [4] PSNR/SSIM	SRCNN [2] PSNR/SSIM	DRCN [4] PSNR/SSIM
Set5	24.69/0.7735	28.42/0.8104	30.48/0.8628	31.53/0.8854
Urban100	20.97/0.6239	23.14/0.6577	24.52/0.7221	25.14/0.7510

Table 5: Metrics from other works



(a) ground truth (b) bicubic (c) SRCNN (d) DRCN

(e) Main, Zero-Pad (f) Upscale only (g) Two-step (h) Deep Net

(i) ground truth (j) bicubic (k) SRCNN (l) DRCN

(m) Main, Zero-Pad (n) Upscale only (o) Two-step (p) Deep Net

Figure 2: Results of some of our approaches, compared to the ground truth, naive interpolation, SRCNN [2] and DRCN [4], both images are from the Set5 dataset [1]. The upper image is image 003, cropped to the lower right. The lower image is image 002, cropped to the upper right. Zero-padding artefacts can be observed in the first three of our images respectively (e, f, g and m, n, o).

7 Discussion

Generally we trained our networks for 500 epochs. The training of the main network with repetition padding had to be aborted at 30 epochs because of external circumstances. From the results of this short training we could already see that it removes the border artefacts as expected. Because of that we did not redo the training even though the overall performance was not on par with the other networks yet.

All other networks were trained under the same circumstances and can be compared. It is fairly obvious, that the network without residual skipping failed completely. In fact, the produced pictures were more or less uniformly gray. Thus we see that residual learning is absolutely necessary for our recurrence. Furthermore this explains the vanishing gradients. Most of the output of one recurrent iteration comes from the previous iteration. Only small corrections are caused by the actual network inside. Network parts with small influence always have small gradients too.

According to the results in Table 3 the network variants generally improved the results as expected. All networks (except for the failed non-residual one) perform better than the bicubic interpolation both according to PSNR and SSIM. They all introduce some artefacts. Because of that, in some cases they look less pleasing than bicubic interpolation even though they produce a sharper image. These artefacts could probably be removed with more training (more epochs and/or bigger data sets). The two step training only performs slightly better than the upscale part without recurrence. This shows once more, that our recurrence is too hard to train. For better performance it should probably have more inner channels. Sadly we could not test this because of hardware limitations and time.

Looking at our deeper network (Deep Net) we see that it converges much slower than the others, which should come as no surprise. After the same number of epochs the network produces results with insufficient coloration (see Figure 2p), the PSNR is rather low and the loss is higher than for the other functioning methods. But against our expectation the SSIM is still higher than for some of the other networks, which suggests that the details where better reconstructed. Regarding this and the fact that the gradient norm is still relatively high, we think that given the appropriate time to train, overall this network would perform the best.

For unknown reasons, our PSNR and SSIM metrics do not match those of other papers (mismatching on bicubic interpolation, see table 5). It is unclear to us why exactly the metrics do not match. We computed the PSNR ourselves and used a reference implementation for SSIM. When comparing the baseline of Kim, Kwon Lee, and Mu Lee [4] to ours, the performance of our networks are acceptable considering the difference in training time (their $\sim$6 days vs our $\sim$1 day).

8 Conclusion

We construct a network for super resolution. In the architecture of this network we combine two approaches, an auto-encoder like structure for upscaling and a recurrent CNN for refinement. We try different variants of the network to study the influence of its aspects.

Our first finding is that using residual learning in the recurrent part is instrumental to obtain reasonable convergence speed. Another observation is that zero-padding in the convolutional layers leads to artefacts at the borders. This can be avoided by introducing replication-padding layers before the convolutions. We also clearly see that deeper networks converge much slower.

Our method shows an improvement compared to several naive upscaling methods (nearest neighbour, bilinear, bicubic), but it is far from reaching state of the art performance. The most obvious reason for this is the lack of training time. Additionally we theorise that a deeper network would work better. But again, this would induce some serious augmentation of the computational complexity.

Improvements to our systems could certainly be achieved by conducting a series of experiments to better tune our hyperparameters. But this was not possible to achieve with our limited resources.

References

[1] Marco Bevilacqua et al. "Low-complexity single-image super-resolution based on non-negative neighbor embedding". In: (2012).

[2] Chao Dong et al. "Image super-resolution using deep convolutional networks". In: *IEEE transactions on pattern analysis and machine intelligence* 38.2 (2016), pp. 295–307.

[3] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. "Single Image Super-Resolution from Transformed Self-Exemplars". In: *IEEE Conference on Computer Vision and Pattern Recognition)*. 2015.

[4] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. "Deeply-recursive convolutional network for image super-resolution". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1637–1645.

[5] D. Martin et al. "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics". In: *Proc. 8th Int'l Conf. Computer Vision*. Vol. 2. 2001, pp. 416–423.

[6] Roman Zeyde, Michael Elad, and Matan Protter. "On single image scale-up using sparse-representations". In: *International conference on curves and surfaces*. Springer. 2010, pp. 711–730.