Josh Sutton and Mick Leungpathomaram

Jerod Weinman

December 14, 2022

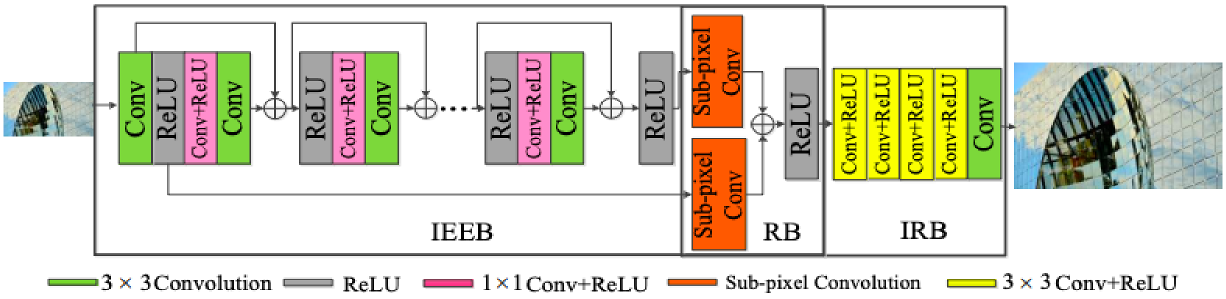
CSC-262

Super-Resolution with Convolutional Neural Networks

Super-Resolution is the process of taking an input image and upscaling it to a higher resolution, ideally with minimal loss to detail and image quality. Convolution Neural Networks(CNNs) have proven to be effective structures in solving Super-Resolution problems. Over the course of the project, we implemented three different Single-Image Super Resolution networks, a simple 3 layer Super-Resolution Convolutional Neural Network (SRCNN) as proposed in [1], the faster and stronger Fast Super-Resolution Convolutional Neural Network (FSRCNN), as proposed in [2], and an even stronger and more complex Lightweight Enhanced SRCNN (LESRCNN), as proposed in [3].

We then tested the networks on our datasets with a variety of hyperparameters, and compiled the results below. Overall, we found that our implementations of all 3 networks were capable of substantial quantitative and qualitative improvements to the images, although they were still inferior to the results in the original papers. The SRCNN and FRSRCNN generally performed as well as or worse than as bicubic interpolation, while the LESRCNN performed as well as or better than bicubic interpolation.While both networks performed roughly the same, the FSRCNN implementation was vastly slower than both the original and the SRCNN, but had a much longer trainer time. Our implementation of LESRCNN consisting of over 936,897 learnable parameters outperforms both the SRCNN and FSRCNN, but has a longer training time of over 40 minutes.

LESRCNN:

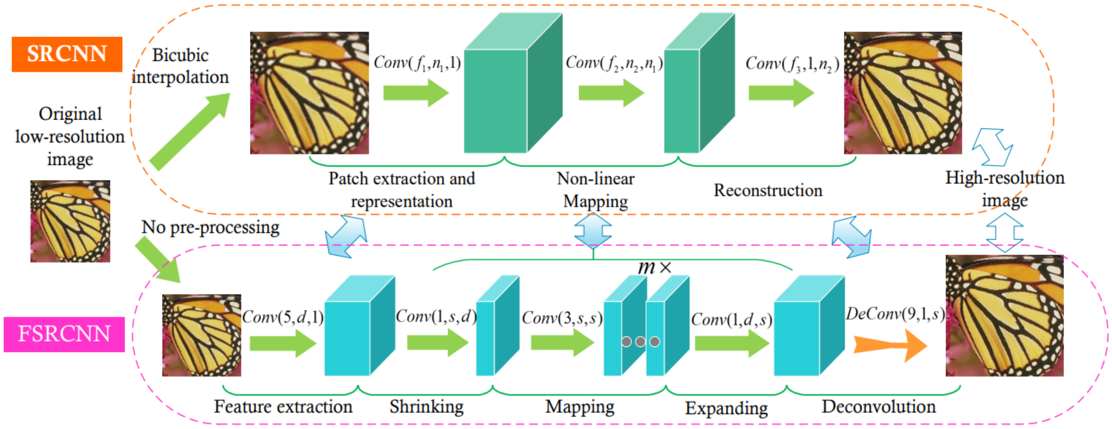


[3] figure 1; LESRCNN Structure

The LESRCNN network consists of three blocks, an “information extraction and enhancement block (IEEB), a reconstruction block (RB), and an information refinement block (IRB)” [3]. The IEEB block extracts and aggregates lower resolution features and uses a residual training method, applying element-wise addition with each odd convolution to keep accurate hierarchical information. Then, the RB block performs residual learning on the output of the IEEB first and last ReLU layer. For our scaling factor *r*, the RB convolves sum of the two layers, expanding the number of channels to be r^2 times larger, creating a (H x W x C*r^2) tensor shaping. Then, this tensor is passed through the shuffle layer outputting an upsampled (rH x rW x C) tensor which blends information from the channels [4]. Finally, the IRB processes the inputted course high frequencies to increase accuracy of the high resolution image with 4 convolution and ReLU layers [3].

One of our main priorities when implementing the LESRCNN was to minimize training time and computation complexity. We decided to decrease the number of layers in the IEEB from 17 to 9, to decrease the number of parameters. Similarly, we reduced the training input patch size from 64 to 12 for most of our testing. While we briefly investigated decreasing the channel dimensions, we found that this parameter was essential to network performance, and despite the fact that our network had far more learnable parameters than before, informal testing indicated it was worth it to keep this constant. Furthermore, Matlab has no built-in shuffle layer. We created a custom shuffle layer in order to take r^2 channels and output a single image with each *r x r* subset of the output high resolution features consisting of the same pixel location of the r^2 input channels.

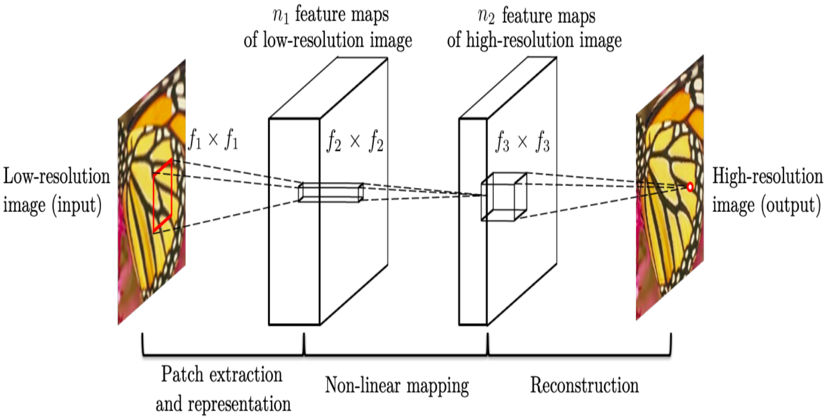
FSRCNN:



[2] figure 2; FSRCNN Structure

The network consists of an initial patch extraction layer, followed by a feature dimension shrinking layer to reduce computation complexity. We next have 2-4 non-linear mapping layers with medium sized(3x3) filters, designed to improve the network. After that, we have an expanding layer, acting as the inverse of the shrinking layer, followed by a transposed convolutional layer which upsamples the data into a HR image. Each layer except for the transpose is followed by a PRELU activation function. To train the network, we randomly extracted 7x7 patches from each image in our training dataset as input into the network. After some experimentation, we found that having only 2 mapping layers works best, and decreases training time substantially.

SRCNN:



[1] figure 3; FSRCNN Structure

The SRCNN consists of 3 convolutional layers, with RELU activation functions after each one, and a regression layer at the end, each with randomized weights. The first convolutional layer is designed to extract patches from the input, representing each as a high-dimensional vector. The second non-linearly maps each vector onto another. The third layer aggregates the representations together, constructing a high-resolution output image. To train the network we randomly extracted [32/64/96/128] 33x33 patches from our training dataset as input to the network.

Evaluation:

For each of the networks, our primary metric was the average Peak-Signal-Noise-Ratio (PSNR) between each of the testing input sub-images and their respective response sub-images. This was compared against the average PSNR when upscaling those sub-images using bicubic interpolation. We also kept track of the runtime, although this was only used to compare between networks.

Throughout the project, we used the Urban100, BSD100, Set14, and Set5 datasets, which are datasets commonly used for Super-Resolution[7]. We also used 100 images from the OST300 dataset, a dataset containing outdoor scenes [8]. Throughout the rest of the paper we refer to this as the OST100 dataset. For training, sub-images are extracted from the training network, with the quantity and size varying depending on the network and hyperparameters. These sub-images are also augmented via 90-degree rotations. Testing involves extracting a single larger sub-image from each of the testing images, to ensure uniform network input.

Each of the networks were tested on 2x and 3x upscaling with Urban100 as their training set, Set5 as the validation set, and OST100, BSD100, and Set14 as testing sets, with initial learning rates of [10^{-3}/10^{-4}/10^{-5}]. All of the networks use MSE as their loss function implementing Adam optimizers and all of the datasets were converted from RGB to single-channel luminance images.

Challenges:

One of the main hurdles of the creation of the project was understanding how to use Matlab's Deep Learning Toolbox to its fullest extent, most of which revolved around the Datastore objects that the Toolbox introduced. We initially attempted to perform Patch Extraction on our ImageDatastores using Matlab's *randomPatchExtractionDatastore* function, but later realized that creating a custom function was easier to both implement and modify. Similarly, the TransformedDataStores created by using the *transform* function to resize the image and change the color channel of images in our ImageDataStores caused incompatibility issues when input into the network. We found it much easier to convert the ImageDatastore into an array of images, which we could then transform and feed into the network with no issues.

Creating the LESRCNN network within Matlab required different training models and convolutional layer formats. Residual training within each odd layer of the IEEB block created difficulties in applying element-wise addition of past convolutions, but we found the additionLayer which would provide necessary operation to perform residual learning methods.

Understanding and implementing the shuffle layer of the sub-pixel convolutional layer was also a difficult challenge within the RB block. We first tried reshaping to change the tensor shaping of our input patches, but this resulted in a recurring image of smaller scale. Finding an implementation of a shuffle layer with a scaling factor of 2 allowed us to fully understand the shuffle layer, and implement our own, more streamlined version for 3x upscaling [5]. Shuffling with a scaling factor of three takes each 9 channels and outputs a single image of 3*H x 3*W x C shaping, with each 3 x 3 subset of the output consisting of the same pixel location of the 9 input channels [4].

Results:

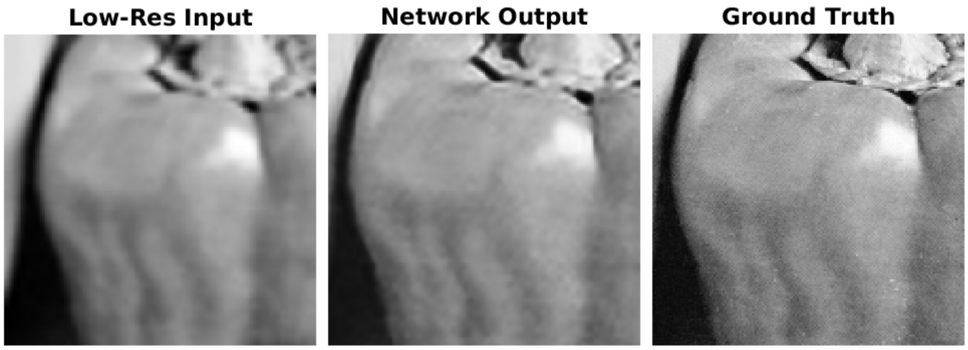


Figure 4; SRCNN with upscaling factor of 3, predicted Set14 image (Avg. PSNR of 26.78)

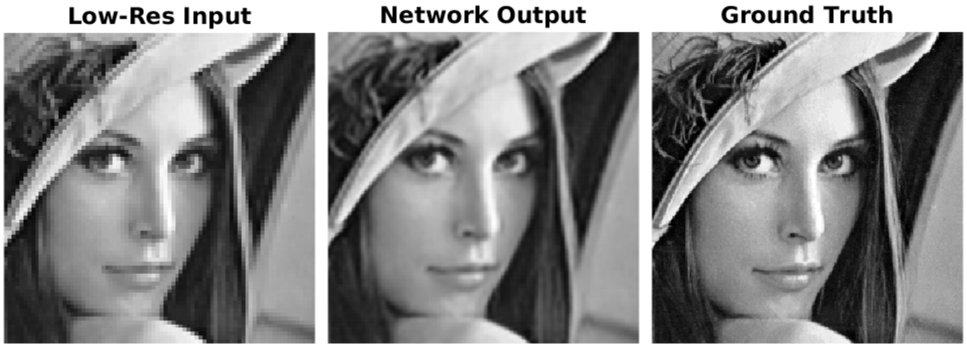


Figure 5; FSRCNN with upscaling factor of 3, predicted Set14 image (Avg. PSNR of 24.39)

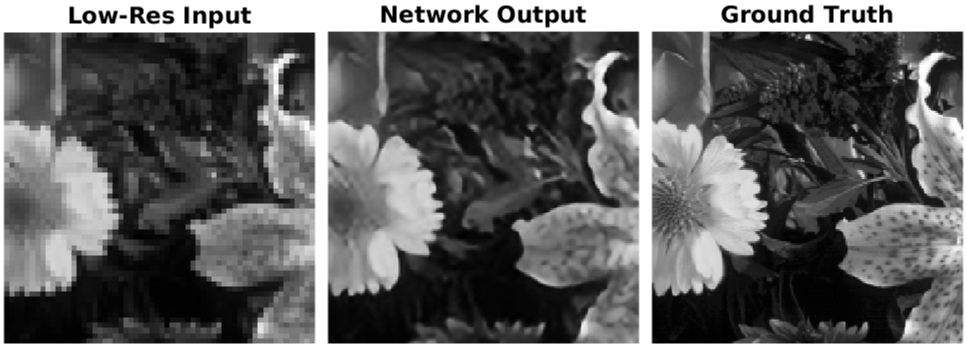


Figure 6; LESRCNN with upscaling factor of 3, predicted Set14 image (Avg. PSNR of 27.95)

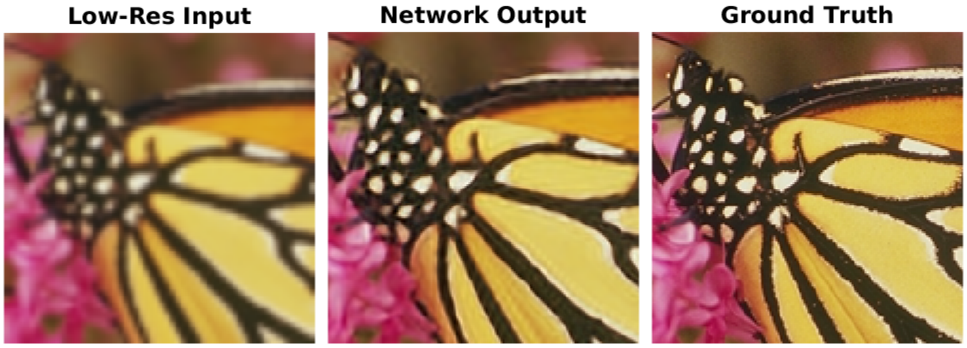


Figure 7; SRCNN with upscaling factor of 3, predicted RGB Set14 image (PSNR of 26.22)

The output bell pepper image in figure 4 has defined details most visible on the stem of the vegetable, decreasing the overall blurring effect of the inputted low resolution image.

Observing the network predicted image of our LESRCNN in figure 6 we can visually see beyond PSNR values that the accuracy of our outputted high resolution image more closely resembles the ground truth. The edges of both the lily on the right and the daisy on the left of the network predicted image are more defined. We can also see that details within the lily are expressed more closely resembling the ground truth image.

Applying the SRCNN, trained with inputted RGB images rather than luminance, in figure 7 we can see that the network predicted image is less blurry than the inputted test image. The details of the wings are sharper as are the white dots found on the body of the butterfly.

SRCNN Epochs: 300 Train : Urban 100 Subimages: 256 Filter Sizes: 9x9, 5x5, 5x5 trainPatchSize: 33		**Initial Learning Rate**		
		10^{-3}	10^{-4}	10^{-5}
Scaling Factor	x2	(BSD100): **28.60** Bicubic: 28.21 (OST100): 28.03 Bicubic: 29.48 (Set14): 29.86 Bicubic: 29.69	(BSD100): 28.58 Bicubic: 28.21 (OST100): **28.65** Bicubic: 29.48 (Set14): **30.16** Bicubic: 29.69	(BSD100): 28.58 Bicubic: 28.21 (OST100): 28.05 Bicubic: 29.48 (Set14): 29.16 Bicubic: 29.69
	x3	(BSD100): **26.24** Bicubic: 27.22 (OST100): **26.19** Bicubic: 26.29 (Set14): **26.78** Bicubic: 26.05	(BSD100): 26.08 Bicubic: 27.22 (OST100): 25.79 Bicubic: 26.29 (Set14): 26.60 Bicubic: 26.05	(BSD100): 25.70 Bicubic: 27.22 (OST100): 25.58 Bicubic: 26.29 (Set14): 25.90 Bicubic: 26.05

FSRCNN Epochs: 300 Train : Urban100 Subimgs/Img: 256 trainSubImgSize: 7x7		**Initial Learning Rate**		
		10^{-3}	10^{-4}	10^{-5}
Scaling Factor	x2	(BSD100): **26.24** Bicubic: 28.21 (OST100): **26.00** Bicubic: 29.48 (Set14): **27.54** Bicubic: 29.69	(BSD100): 25.50 Bicubic: 28.21 (OST100): 25.51 Bicubic: 29.48 (Set14): 26.00 Bicubic: 29.69	(BSD100): 23.90 Bicubic: 28.21 (OST100): 23.71 Bicubic: 29.48 (Set14): 23.02 Bicubic: 29.69
	x3	(BSD100): 23.80 Bicubic: 27.22 (OST100): **24.02** Bicubic: 26.29 (Set14): **24.39** Bicubic: 26.05	(BSD100): **23.84** Bicubic: 27.22 (OST100): 23.87 Bicubic: 26.29 (Set14): 24.22 Bicubic: 26.05	(BSD100): 23.22 Bicubic: 27.22 (OST100): 23.63 Bicubic: 26.29 (Set14): 22.26 Bicubic: 26.05

FSRCNN Epochs: 150 Train : Urban100 Subimgs/Img: 128 trainSubImgSize: 7x7		**Initial Learning Rate**	
		10^{-3}	10^{-4}
Scaling Factor	x2	(BSD100): **28.58** Bicubic: 28.21 (OST100): **28.16** Bicubic: 29.48 (Set14): **31.23** Bicubic: 29.69	(BSD100): 23.42 Bicubic: 28.21 (OST100): 23.40 Bicubic: 29.48 (Set14): 23.81 Bicubic: 29.69
	x3	(BSD100): **25.87** Bicubic: 27.22 (OST100): **26.12** Bicubic: 26.29 (Set14): **26.59** Bicubic: 26.05	(BSD100): 25.16 Bicubic: 27.22 (OST100): 25.32 Bicubic: 26.29 (Set14): 26.43 Bicubic: 26.05

FSRCNN Epochs: 150 Train : Urban100 Subimgs/Img: 128 trainSubImgSize: 7x7 scalingFactor: 3		**Initial Learning Rate**	
		10^{-3}	10^{-4}
Mapping Layers	2	(BSD100): **25.69** Bicubic: 27.22 (OST100): **26.07** Bicubic: 26.29 (Set14): **27.76** Bicubic: 26.05	(BSD100): 25.45 Bicubic: 27.22 (OST100): 25.53 Bicubic: 26.29 (Set14): 25.89 Bicubic: 26.05
	3	(BSD100): 25.61 Bicubic: 27.22 (OST100): 26.10 Bicubic: 26.29 (Set14): 26.73 Bicubic: 26.05	(BSD100): 24.07 Bicubic: 27.22 (OST100): 24.21 Bicubic: 26.29 (Set14): 24.33 Bicubic: 26.05
	4	(BSD100): 25.44 Bicubic: 27.22 (OST100): 25.88 Bicubic: 26.29 (Set14): 27.23 Bicubic: 26.05	(BSD100): 20.83 Bicubic: 27.22 (OST100): 20.82 Bicubic: 26.29 (Set14): 21.06 Bicubic: 26.05

LESRCNN Epochs: 6 Train : Urban 100 scalingFactor = 3 outputChannels = 64 SubImgs/Img=128 SubImgSize = 12x12		**Initial Learning Rate**		
		10^{-3}	10^{-4}	10^{-5}
Scaling Factor	x2	BSD100: 28.84 Bicubic:28.21 OST100: **28.52** Bicubic:29.48 Set14: **31.66** Bicubic: 29.69	BSD100: **29.00** Bicubic:28.21 OST100: 28.27 Bicubic:29.48 Set14: 31.14 Bicubic: 29.69	BSD100: 27.05 Bicubic:28.21 OST100: 26.81 Bicubic:29.48 Set14: 27.47 Bicubic: 29.69
	x3	BSD100: **26.15** Bicubic:27.22 OST100: **26.52** Bicubic: 26.29 Set14: **27.95** Bicubic: 26.05	BSD100: 26.11 Bicubic:27.22 OST100: 26.40 Bicubic: 26.29 Set14: 27.24 Bicubic: 26.05	BSD100: 25.00 Bicubic:27.22 OST100: 25.19 Bicubic: 26.29 Set14: 25.50 Bicubic: 26.05

RGB Testing(Using Best Parameters Setting for each Network, on 3x upscaling)			
Network	SRCNN	FSRCNN	LESRCNN
PSNR	BSD100: **24.53** Bicubic: 25.9 OST100: **25.70** Bicubic:24.925 Set14: 26.22 Bicubic: 24.4	BSD100: 22.37 Bicubic: 25.9 OST100: 22.46 Bicubic:24.925 Set14: 22.32 Bicubic: 24.4	BSD100: **24.84** Bicubic: 25.9 OST100: **24.78** Bicubic:24.925 Set14: **26.58** Bicubic: 24.4

The constant variables were decided through preliminary testing, to roughly narrow down the ideal range of values, due to limitations in testing time.

Analysis:

Compared to the results in our progress report, we trained the networks with a larger dataset and augmented the datasets with rotated images to prevent overfitting. We also tried different upscaling factors, and tested the images on more datasets.

Our implementation SRCNN performs fairly well, with marked visible improvements to the luminance values in features and feature edges compared to the network input. While running for 60,000 iterations, depending on the number of patches per image, the network produced meaningful results. In contrast, the network in the paper ran for over $12*10^8$ iterations. [1] also had a significantly larger training set, as they extracted 24800 sub-images from their 91 image training dataset, averaging out to roughly 272 patches/img. Compared with our progress report, the SRCNN performance improved drastically as a result of the augmented datasets and the fact that we changed the second filter to be 5x5 instead of 1x1, with the best results exceeding the bi-cubic baseline on both upscaling factors. In its best case, the network exceeds the bi-cubic baseline by 0.73 decibels, with a PSNR of 26.78 dB as opposed to 26.05 dB. While our implementation did not reach the PSNR value of 32.83 as seen in the paper, its results were still fairly respectable given the differences in circumstances.

Our implementation of the FSRCNN is significantly less accurate than our implemented SRCNN, and generally inferior to the bi-cubic baseline and the network as seen in [2]. With *m=2, m=3, m=4,* the actual FSRCNN is expected to have PSNR values of 32.87, 32.88, and 33.08, respectively. This could be in part due to the vastly reduced training time. While we ran our network for 7500 iterations, in [2] the network was trained for upwards of $12*10^8$ iterations. While [2] does not specify the number of patches per image they create, it’s likely that they had a similarly larger training set than ours. Furthermore, network performance scaled inversely with the size of the training dataset and the number of mapping layers. The PSNR drops by roughly 2 points when doubling the amount of training images on both learning rates. We assume part of this is due to the transposed convolution layer, which creates visible filter marks on learned network outputs, as seen below:

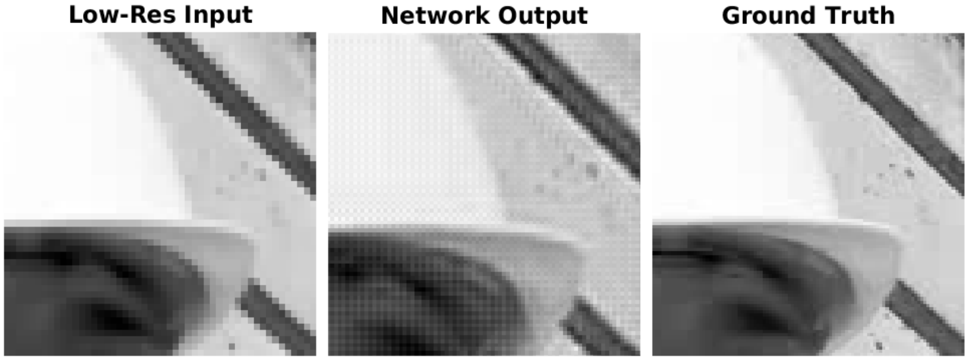


Figure 8; FSRCNN with upscaling factor of 3, predicted image displaying checkerboard artifacts

We suspect that this layer is attempting to invent high frequency details everywhere in the outputs, causing the checkerboard patterning throughout the image, but especially in low frequency areas. With differences of implementation, the original paper avoids this issue.

We also see a small yet consistent decline in performance as mapping layers increase on a 10^{-3} learning rate, and a substantially larger drop on 10^{-4}. However, we can see that the best performance of the network exceeds the bi-cubic baseline by 0.54 decibels, with a PSNR of 26.59 dB compared to 26.05 dB.

Our implementation of the LESRCNN performs better than our SRCNN by a slight yet consistent margin. The average PSNR values found within the paper were 33.93 for an upscaling factor of 3, while we have a peak result of 27.95 dB when testing on Set14 [3]. Our network trains for 15,360 iterations, which compared to our other networks is a marked decrease, but the time for iterations increased sharply due to the number of parameters, leading to an overall substantial increase in training time. At its best, the network exceeds the bi-cubic baseline by 1.9 decibels, with a PSNR of 27.95 dB compared to 26.05 dB.

Overall, all of our networks and the bi-cubic baselines performed better on 2x upscaling than 3x times. This makes sense, as the input images are closer to the output compared to 3x upscaling, decreasing the difficulty of the task.

Both our networks, SRCNN and FSRCNN, closely resemble the total counts found within [2] ours having 57,281 and 11,737 respectively, while networks within the papers have 57,184 [2] and 12,464 [2] parameters. This discrepancy can be a result of differences in platforms and libraries, noting that the original papers were implemented on Pytorch, while our networks were implemented in Matlab using the Deep Learning Toolbox. However our implementation of LESRCNN had 936,987 parameters while the paper states the network has 516,000 [3] parameters. This discrepancy is not only due to changes in hyperparameters as well as layer amounts, but could also be a result of our initialization of the sub-pixel convolution layer, which carries the most parameters in the network, with each layer containing over 300,000 parameters.

References

[1] Dong, Chao, Chen Change Loy, Kaiming He, Xiaoou Tang. ‘Image Super-Resolution Using Deep Convolutional Networks’. arXiv, 2015. https://doi.org/10.48550/ARXIV.1501.00092

[2] Dong, Chao, Chen Change Loy, Xiaoou Tang. ‘Accelerating the Super-Resolution Convolutional Neural Network’. arXiv, 2016. https://doi.org/10.48550/ARXIV.1608.00367

[3] Tian, Chunwei, Ruibin Zhuge, Zhihao Wu, Yong Xu, Wangmeng Zuo, Chen Chen, Chia-Wen Lin. ‘Lightweight image super-resolution with enhanced CNN’. arXiv, 2020. https://doi.org/10.48550/ARXIV.2007.04344.

[4] Shi, W., Caballero, J., Huszar, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., Wang, Z., 2016. ‘Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network.’ arXiv, 2016. https://arxiv.org/pdf/1609.05158.pdf

[5] manoreken. *SRGAN-VGG54 Single Image Super Resolution Matlab port.* MathWorks,2022. https://www.mathworks.com/matlabcentral/fileexchange/95228-srgan-vgg54-single-image-super-resolution-matlab-port

[6] Wang, Xintao, Ke Yu, Chao Dong, and Chen Change Loy. ‘Recovering Realistic Texture in Image Super-Resolution by Deep Spatial Feature Transform’. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[7] Huang, Jia-Bin, Abhishek Singh, and Narendra Ahuja. ‘Single Image Super-Resolution From Transformed Self-Exemplars’. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 5197–5206, 2015.

[8] Xintao Wang, Ke Yu, Chao Dong, and Chen Change Loy. 2018. ‘Recovering realistic texture in image super-resolution by deep spatial feature transform’. In Proceedings of the IEEE conference on computer vision and pattern recognition. 606–615, 2018.