Super Resolution Convolutional Neural Network

Abhishek Jindal aj2708, Jonathan Chan jc4659, Himani Arora ha2434
Columbia University

Abstract

Image super resolution aims at recovering a high resolution image from a low resolution input image by learning a mapping between low resolution and high resolution image spaces.In this paper, firstly, we aim to reproduce and improve upon the results of the SRCNN. We tried out various optimization techniques like batch normalization and regularization to achieve a higher performance. Secondly, we attempt to redesign SRCNN and implement modifications in order to improve the quality of restored images at a lower computational cost. We try to implement the FSRCNN which is a variant of the SRCNN having a deconvolution layer instead of upsampling. Thirdly, we explore the usage of perceptual loss as the cost function instead of the mean squared error loss in both SRCNN and FSRCNN. The main challenge we faced in these tasks was the lack of some specific implementation details in the literature about subsampling, upsampling and deconvolution. Overall, we successfully outperform the SRCNN, achieve state of the art results with FSRCNN and infer that the VGG loss or perceptual loss serves as a very good cost function.

1. Introduction

Super resolution networks seek to recover high resolution images from a given low resolution image. SRCNN is a specific method for super resolution that is patch based and learns the mapping from the low dimensional image space to the high dimensional image space.SRCNN is much faster than other state of the art systems and has a rather simplistic architecture.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

Dong et al. set out to demonstrate that the super resolution task can be accomplished successfully using convolutional neural networks [2]. To this end, they take three stages of traditional super resolution methods and design three convolutional layer equivalents: patch extraction, nonlinear mapping, and reconstruction.

To replace the patch extraction stage, Dong et al. design the first layer as a convolution that uses ReLu as the activation function. The goal of this layer is to build a set of filters that replaces the pre-trained bases, usually PCA, used in traditional methods. The second layer is also a convolution with ReLu that is used to replace the traditional non-linear mapping phase. This layer simply maps the input to a different dimensional space. The third and final layer replaces the reconstruction stage by using convolution with a linear function, where the goal is to have the filters project the image represented in a different dimensional space back into the image domain and to have the filters perform averaging. Figure 1. describes this architecture in graphical form.

To train the network, the authors use mean squared error as the cost function. Because of the downsampling that occurs with the network, the mean squared error is calculated between the center pixels of the ground truth and the network output.

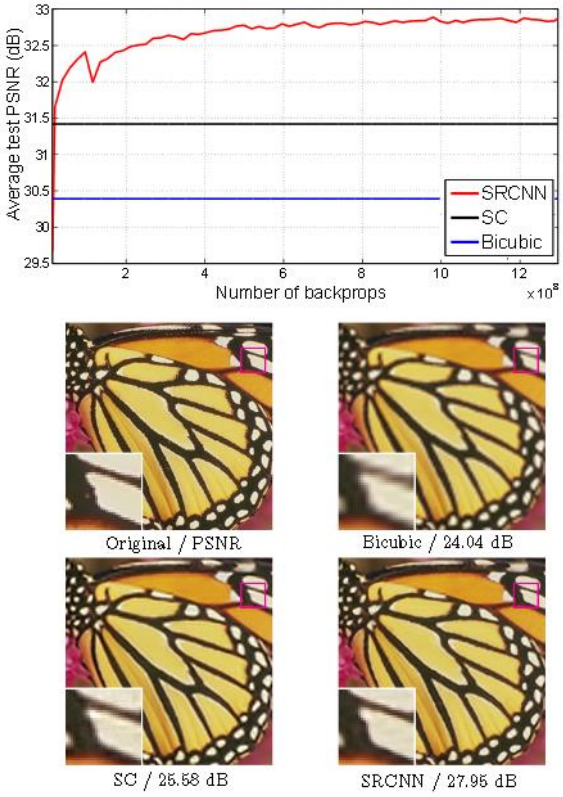
For their experiments, they use the canonical 91 image dataset for training, Set 5 for validation, and Set 14 for testing. The inputs to the network are the ground truth images, divided into subimages, Gaussian blurred, subsampled by the intended upscaling factor, and then upscaled with the same factor using bicubic interpolation.

In addition to the original paper, we did a thorough literature survey of other papers on the subject of super resolution for images. One such paper was the Fast Super Resolution Convolutional Neural Network in which the authors describe a variant of the SRCNN(referred henceforth as the FSRCNN). The key objective of the FSRCNN is to bring about a massive speed up (nearly 41 times) compared to SRCNN(so as to make the network realtime) without loss of accuracy - in fact FSRCNN improves marginally over SRCNN's performance.To save the quadratic cost of bicubic interpolation, FSRCNN upsamples by using a deconvolution layer as the last layer. The deconvolution layer dynamically learns upsampling kernels that work to generate the final high resolution output. Therefore, the FSRCNN does feature extraction on the original The second important optimization which the FSRCNN applies is the use of a shrinking layer at the beginning and an expanding layer at the end to restrict mapping in a low dimensional feature space. The overall network has a symmetrical hourglass shape (thin in the middle, thick at the ends) and has fewer parameters than the original SRCNN.

It has been mentioned that per pixel mean squared error and PSNR are only weakly correlated with the quality of the reconstructed high resolution images. Therefore,

many other recent authors such as Johnson et al. and Ledig et al. have replaced the general per-pixel loss function with perceptual or content loss in the context of image super resolution. Such loss functions measure the perceptual similarity between two images and are able to capture a higher level of information about the image. They are based on the idea that we look for a greater number of details in a super resolution image than just pixel-wise similarity to its low resolution counterpart. Thus, the SR images based on the perceptual loss have been shown to be more visually appealing in the original papers. It is calculated after transforming the image in the feature space of the pre-trained VGG network described by Simonyan et al..

2.2 Key Results of the Original Paper



In the original paper, the super resolution is achieved by first upsampling the low **resolution path using** bicubic interpolation and then that patch is passed through SRCNN as the number of backpropagations through the network increases. It is observed that SRCNN has a much higher test PSNR compared to standard techniques like Bicubic. SRCNN has a very simple architecture compared to other state of the art systems for super resolution. Also, in the original paper, authors experiment with different channels and report their best results with the luminance channel.

3. Methodology

We first describe the objective of our project along with the technical challenges faced by us. More detailed architectural and experimental details are described in section 4.

3.1. Objectives and Technical Challenges

The objective of our project was to try and emulate the results of the SRCNN and experiment with various deep learning concepts in achieve better performance than the SRCNN. We also wanted to learn other networks for super resolution like the FSRCNN and experiment with different cost functions.

We faced quite a lot of technical challenges in our project. It was not trivial to implement the deconvolution layer (considering that Theano has no inbuilt function for it). Also, in absence of specific details about reconstruction and subsampling, it was hard for us to implement it correctly. In fact, it was only a few hours before the submission deadline we realized a small bug in our subsampling code which was ruining the quality of the results.Therefore, some of the old experiments run on jupyter notebook and the outputs saved in our project repository couldn't be updated with the latest findings.

3.2. Problem Formulation and Design

In this paper we present the problem of generating super-resolution images from the given low-resolution images.

We attempted to build the SRCNN as presented by Dong et al. We trained and tested the network on the same dataset. In addition we experimented with different hyperparameters and optimizations techniques.Then we experimented with different fast super resolution convolutional networks in order to avoid the computational complexity associated with bicubic interpolation and to achieve some performance gain over the original CNN.Taking inspiration from Johnson et al. and Ledig et al, we also incorporated the perceptual loss function in our original network**.** Furthermore, we tried to integrate the concept of perceptual loss function in the variants of the FSRCNN in order to improve our network's performance.

4. Implementation

We first describe the architecture of the SRCNN, followed by the architecture of FSRCNN, followed by the concept of perceptual loss and how we utilized this concept to improve the performance of our network.

4.1. Deep Learning Network

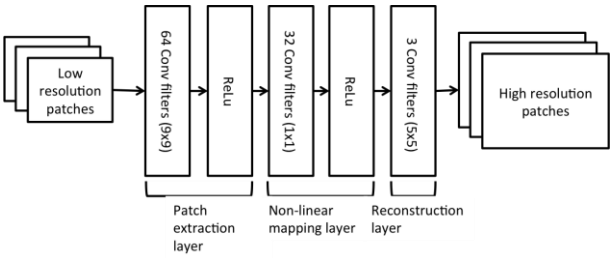
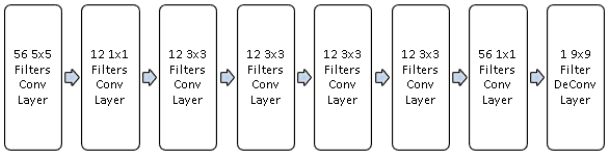


Figure 1. SRCNN Architecture. From left to right, data in the form of low resolution patches are passed through the patch extraction layer, non-linear mapping layer, and the reconstruction layer to create high resolution patches.

The intuition and motivation behind the SRCNN architecture is described in Section 2. For the SRCNN network there are a number of options regarding filter sizes and number of layers. Specifically, we chose the patch extraction layer that uses 64 convolution filters that are 9x9 in width and height followed by the ReLu activation function. The non-linear mapping layer uses 32 1x1 convolutional filters followed by ReLu. The reconstruction layer uses 3 convolutional filters that are 5x5. In addition, we also rescale the pixel values between 0 and 1 and clip the output of the network to 0 and 1 to prevent any sudden spikes when adjusting the network weights during training.

Our implementation of SRCNN was greatly helped by Theano tutorials [4,5,6]. We experimented with a variety of optimization algorithms: Adam, RMSProp, and SGD. We report the results of these experiments in Section 5. Dong et al. use SGD with fixed learning rates for the first two layers and a different learning rate for the last layer.

We used the same datasets as mentioned in Section 2 to train, validate, and test our architectures. When running the experiments, the paper states that they created 24,800 subimages that are 33x33 extracted from the original images with a stride of 14. In our experiments, we created 22,092 subimages of size 33x33 with a stride of 14. We also prepare the low resolution patches by Gaussian blurring with a sigma of 1; it is not stated how much blurring the original authors use.



We also implemented several variants of the Fast Super Resolution Convolutional Neural Network(illustrated above). In FSRCNN, the low resolution image is directly input to the network (without any interpolation). It is then passed through a mapping layer with 56 filters of size 5x5. The mapping layer is followed by the shrinking layer of 12 filters of 1 x1 to reduce the feature space low resolution feature dimension and the number of parameters. Then it is followed by a stack of 4 convolutional layers having 12 filters of 3 x3 to act as the non-linear mapping layers. Then a convolution expanding layer with 56 1x1 filters acts like the inverse of shrinking layer at the start of the network. Finally the last layer is a deconvolution layer which learns the upsampling kernels and has 1 9x9 filter.The output of the network is the upsampled image which has output size equal to the size of the input low resolution image divided by the stride size of the deconvolution layer. The FSRCNN uses the Parameterized Rectified Linear Unit (PReLU) as the activation function at each convolutional layer and it learns the leak coefficient is learned by the network through backpropagation.

In order to measure the perceptual loss, we passed both the original high resolution image as well as the output from our implementation of SRCNN into the VGG network loaded with pre-trained weights. These weights, trained on the ImageNet dataset, have been made available by the original authors. Following in the steps of Johnson et al., we used the 16 layer VGG network which is a deep convolutional network and uses 3x3 kernels throughout with a fixed stride of 1 [7]. The images are zero-padded such that the output of a convolutional layer has same size as the input to that layer. All the convolutional layers use ReLU non-linearity and only some of them are followed by 2x2 max pooling with a

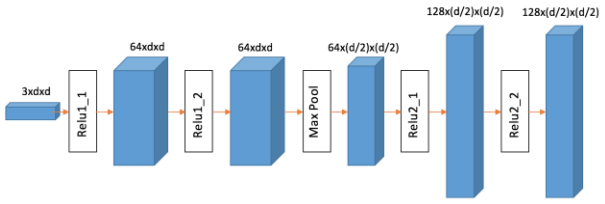


Figure 2. The first 4 layers of the VGG-16 Architecture. The image dimensions are shown above each layer. The input image is taken from the output of the SRCNN layer and has 3 channels. The output of the VGG networks has 128 channels. 'Relux_y' refers to the convolutional layer with Relu activation function.

From left to right, data in the form of low resolution patches are passed through the patch extraction layer, non-linear mapping layer, and the reconstruction layer to create high resolution patches.

stride of 2. Some fully connected layers follow the convolutional layers in the end, however, since we were interested in only the first few convolutional layers, we have not described them. Again, to match with Johnson et al., we took the output from 'relu2_2' (see Figure). Since our original high resolution image and the output of SRCNN had different dimensions, we used only the central part of the image to calculate the perceptual loss.

5. Results

5.1. Project Results

Figure 3. The results of the VGG experiments in Section 5.1.1. From left to right: using only VGG loss, using the sum of VGG loss and SRCNN MSE loss and trained for 20 epochs, using the sum of VGG loss and SRCNN MSE loss and trained for 50 epochs, using a lower learning rate for last convolutional layer. These images have been taken from the test dataset 'Set14'.

5.1. 1 SRCNN Results

To test the SRCNN network, we generated subsample images by extracting 33x33 patches in RGB images with a stride 14. We blurred the image, used bicubic interpolation to downsample by a factor of 4, and upsampled to get the low resolution patch. The network was trained on all 3 channels, and results are reported on the test set known as Set 14.

We experimented with 3 different optimization algorithms: Stochastic gradient descent (SGD), Adam, and RMSProp. For SGD, we followed Dong et al. in assigning the first two layers a learning rate of 1e-4 and the last layer a learning rate of 1e-5. Strangely, we were not able to get reasonable performance with this arrangement. We introduced Adam and RMSProp to train the network with significantly better results, reported in Table 1. We also noticed that the network appeared to have difficulty passing a plateau in the cost function, so we experimented with adding batch normalization[6]. The hypothesis was that batch normalization should make the easier to train ReLu, which are prone to poor initialization. These results also show counter-intuitively that batch normalization did not improve the PSNR. During training, we saw that the training PSNR diverged from the validation and test PSNR, meaning that the network was overfitting. We introduced an L2 norm of 1e-4 after some hyperparameter searching, which indeed improved the SRCNN network performance.

Experiment	**SGD**	**Adam**	**RMSProp**
Test PSNR	6.51	35.08	34.05

Experiment	**RMSProp w/o L2 norm**	**RMSProp w/ L2 norm**
Test PSNR	34.05	34.29

Experiment	**Batch Normalization + L2 Norm**	**Without Batch Normalization + L2 Norm**
Test PSNR	14.0	34.29

Table 1: The above tables describe 3 experiments that were run on the SRCNN architecture, showing that RMSProp with L2 norm provided the best PSNR.

Comparing the results in Set5, which was our validation set, the PSNR for RGB images upsampled at 4x was 36.5 around 44,000 iterations with Adam using minibatches of size 20, compared to 30.48 based on Johnson's implementation of SRCNN (the original authors did not test RGB at 4x upscaling); on Set 14, our test set, the results are equally stark 27.49 compared to 35.08 [7]. Given that 100M iterations were required for 3x upsampling, it is not unreasonable to expect that it would have taken as long to achieve 30 PSNR. From our experiments, SGD decreased by thousandths of a unit in PSNR per epoch. Some of the major differences are that we have the benefit of using a smarter init with Xavier initialization for ReLu, which is an activation that is known to be finicky with the initialization. We also use

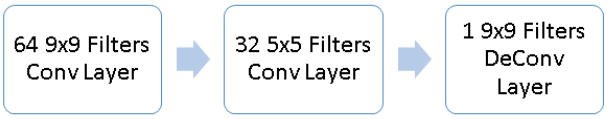
a 2nd order optimization function that is likely to converge faster.

	SRCNN implemented by Johnson[7]	**Ours**
Set 5 PSNR	30.48	36.5
Set 14 PSNR	27.49	35.08

Table 2: The above table shows how the PSNR in our implementation of SRCNN is significantly higher than the original paper. This is likely due to better convergence from 2nd order optimization algorithms and better initializations of ReLu.

5.1.2 FSRCNN Results

For supplying low resolution patch images to the FSRCNN, we did subsampling by creating 8 x 8 patches in RGB. As before, the network was trained on all three channels and the validation was carried on the dataset called Set5 and the testing was carried out on the dataset called Set14. The original paper only provides architecture for the network for a single channel (the Y channel), so therefore we had to modify the network to handle RGB images.
We implemented several variants of the network.



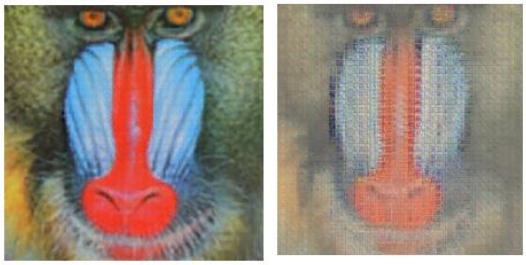
1.Initally, we implemented a concise and a simplified version of the FSRCNN as a proof of concept for actually verifying the performance of the FSRCNN compared to SRCNN. For this network, we experimented with the number of filters in the deconvolution layer and the activation function for the convolutional layers in this network.
We observed that the performance of the network was drastically network when the filter size in the last layer was symmetrical(9).We tried out Relu and Parameterized Relu.We observed that up till 100 epochs, Parameterized Relu didn't offer any significant gain over Relu and in fact, was much slower to train.
We observed that the Half-FSRCNN had some degree of overfitting and we therefore, tried randomly flipping and randomly translating each mini-batch during training. This resulted in a small gain in terms of PSNR and the quality of reconstructed images.

2.Architecture implemented for the full FSRCNN described in section 3. In addition to repeating the experiments carried out for half FSRCNN, we experimented by adding skip connections between different layers in order to improve the results. We also tried out different upsampling Surprisingly, we found that this model was much harder to train.

3.We tried out various implementations of the deconvolution layer. We tried out a naive implementation using border_mode = 'full' in the conv_2d API in order to upsample the input by the filter size.While this particular implementation trained fast, the results of this upsampling process were not great. Then we tried out the theano function AbstractConv2d_gradInput. For implementation the deconvolution, we observed that for most variants of the FSRCNN, AbstractConv2d_gradInput didn't start producing decent reconstructions till at least 200 epochs.

4.We experimented with 2 different optimizers for this case :- Adam and RMS Prop and we observed that Adam gave much better results (in terms of a higher PSNR value on the test set) and converged much faster. However, for some set of values of hyperparameters, Adam converged way too quickly and in these cases, RMS Prop sometimes achieved better values of the objective function.
5.We also experimented with various learning rates and batch sizes.We observed that smaller batch sizes led to significantly better performance of the network.We varied batch sizes from 20 to 500 and observed that for various configurations of the FSRCNN, the optimal value of the batch size varied from 20 to 50. In terms of learning rates, we observed that only very small learning rates (0.001 to 0.005) gave good results. For improving the quality of the results, we tried a small decay parameter for the learning rate in addition to the Adam optimizer (which we typically set to a value just smaller than one - 0.9975 or something similar) and this resulted in marginally better performance. We also did some experimentation with the objective function by trying out different ways to calculate the mean squared error(separately for all channels or together for all channels).
Due to limitation of space, we can't report all our experiments for this section, however they can be found in our project source repository.

	FSRCNN Transition State 1 Original Paper	**Ours - Modified FSRCNN**
PSNR on Test Data	31.95(10000+epochs)	31.75(after 100 epochs)
PSNR Results with PRelu	32.95(10000+epochs)	32.29(after 100 epochs)
PSNR Results with Data Augmentation	Not Reported	23.34 (After 20 epochs) **34.74** (After 100 epochs)



The images above the outputs of the first image generated after 50 epochs - the first image corresponds to Relu as the activation unit and the second image corresponds to the Prelu. Clearly, after 50 epochs the output of the Relu activation is much better .

5.1. 3 VGG Results

In all the experiments with VGG, we used the a learning rate of 0.0001 and a batch size of 100 together with Adams. The experiments 1 to 4 have been conducted on the initial version of upsampled images which was later on improved. The later experiments used the newer version of upsampled images. However, we feel that the same trend as demonstrated in the initial experiments will also hold for them.

1. We used a new loss function given by the VGG loss which essentially calculates the MSE over the images in VGG space when the output is taken from 'Relu2_2'. The final test images did not look as visually appealing as expected and had low contrast regions. In order to compare our results with the SRCNN results, in addition to the PSNR in the VGG space, we also calculated the PSNR in the original space which turned out to be lower.

2. Then we used the sum of VGG loss and the original SRCNN loss together. We saw that the image after the same number of epochs looked better and had a higher PSNR both in the VGG space and the original space as compared to using just the VGG loss alone. To see if this improves the results, we ran it for a larger number of epochs. We saw that the PSNR did improve but only by a small amount. Another aspect that caught our attention was that that after about 20 epochs the PSNR kept oscillating and thus, remained nearly constant.

3. To address the oscillating PSNR and MSE in the previous experiment, we used a lower learning rate of 0.00001 in the last convolutional layer as suggested by Dong et al. This definitely improved the final reconstructed image which was apparent from the higher PSNR, lower MSE as well as the image itself which had more saturation and looked better overall.

Fig: The results of VGG experiment conducted on new upsampled image with an upsampling factor of 4. From left to right: original high resolution image, low resolution image, reconstructed high resolution image.

4. We also tested the VGG network on the improved subsampled images which showed a significant jump in the PSNR value. This had all the best parameters from experiments 1-4 incorporated in it. We ran this network setting twice, once for 20 epochs and the other for 40 and saw that it in the first epoch it achieved a PSNR of 27.19 which increased steadily in the first few epochs, which we believe owes to the fast convergence of Adam. After 10 epochs, although the PSNR value continued to rise, the increase became much slower. It is worth mentioning that after 50 epochs was down to 1.83 (see Figure above for comparison).

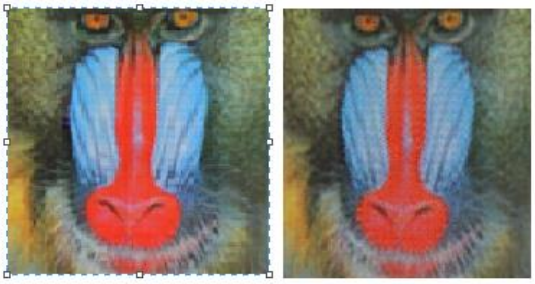
Experiment	**Test PSNR (orig)**	**Test PSNR (VGG)**
Only VGG loss	15.93	20.00
VGG loss + SRCNN loss (20 epoch)	16.09	19.96

VGG loss + SRCNN loss (50 epochs)	16.77	20.22
Lower learning rate in last layer	16.90	20.23
Improved upsampling (20 epochs)	33.32	45.51
Improved upsampling (40 epochs)	**34.23**	46.75

Table 2: The Test PSNR in the original and VGG space for the experiments with VGG network as explained in section 5.1.1

5.1.4 FSRCNN with Perceptual Loss Results

On observing the remarkable effectiveness of the VGG loss, we decided to integrate this concept with the FSRCNN in hope of observing some improvement in performance. We tried changing the cost function of the original FSRCNN to the VGG loss and also tried out combinations of the mean squared error and the VGG loss. We found that VGG loss performed better and the combination of MSE + VGG loss performed even better. In the table below, the first row shows the original PSNR, whereas the second and the third rows describe the modified PSNR which has been calculated with the updated cost function(therefore is on a different scale).However, in terms of quality of reconstructions we could see the superiority of perceptual loss.



The first image above is the output with VGG loss + MSE as the objective function for FSRCNN. The second image above is FSRCNN with MSE as the cost function. Clearly, it can be seen that the resolution of the first image is quite superior to that of the second.

Test Metric	**Our Modified FSRCNN**
PSNR on Test Data	31.75(after 100 epochs)
Modified PSNR with VGG loss	20.49(after 100 epochs)
Modified PSNR with VGG Loss + MSE	20.55(after 100 epochs)

6. Conclusion

We implement and improve upon the existing results of the SRCNN. We achieve nearly state of the art results with FSRCNN using using just 100 epochs. We experiment with VGG loss as our cost function and observe significant improvement in the reconstructed images. In the future, we would like to improve further upon our current experimentation with other super resolution networks and .loss functions to realize the goal of super resolution.

6. Acknowledgement

We didn't take any explicit help for this project from anyone but we are forever thankful to Professor Zoran Kostic and the teaching assistants for their constant motivation, help and guidance.

7. References

Include all references - papers, code, links, books.

[1] Project repo: https://bitbucket.org/e_4040_ta/e4040_project_ahj1/overview

[2] Dong, Chao, et al. "Learning a deep convolutional network for image super-resolution." *European Conference on Computer Vision*. Springer International Publishing, 2014.

[3] "Classifying MNIST digits using Logistic Regression." [Online]. Available: http://deeplearning.net/tutorial/logreg.html

[4] "Multilayer Perceptron." [Online]. Available: http://deeplearning.net/tutorial/mlp.html

[5] "Convolutional Neural Networks (LeNet) [Online]. Available: http://deeplearning.net/tutorial/lenet.html

[6] D. Hjelm. (2016,Nov 3). "Is there anyone who have used 'theano.tensor.nnet.bn.batch_normalization(inputs, gamma, beta, mean, std, mode='low_mem')' [Online]. Available: https://groups.google.com/forum/#!topic/theano-users/dMV6aabL1Ds

[7] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." *arXiv preprint arXiv:1603.08155* (2016).
[8]Chao Dong,Chen Change Loy ,Xiaoou Tang. "Accelerating the Super-Resolution Convolutional Neural Network".