Image Super-Resolution using Deep Learning Methods

Kunal Suthar[1], Maulik Limbadiya[2]

Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar

[1]`201401131@daiict.ac.in`

[2]`201401189@daiict.ac.in`

Supervisor
Prof. Manjunath V. Joshi

Abstract – Single Image super-resolution(SR) problem of the Computer Vision paradigm, can be attempted to be solved by many approaches like Motion-based SR, Zoom-based SR, Blur-based SR and Learning-based SR[3]. We attempt to solve the Single Image SR problem by using deep-learning methods. In this method, we employ and train a deep convolutional neural network(CNN) to learn the patch correspondence between the set of low-resolution images and the high-resolution images. Therefore by using the trained neural network model, we obtain magnification factors of 2x, 4x and 8x, with minimal blur and minimal loss in image quality and details. We further improve the image quality by optimizing the loss function of the Convolutional Neural Network, by adding the regularization term of a smoothness prior, in the loss function.

Keywords — Super-Resolution, Deep Convolutional Neural Networks, Machine Learning, Regularization, Patch-Extraction, Non-Linear Mapping, Image reconstruction

I. Introduction

Single image super-resolution is the method for constructing a High-resolution(HR) image from a Low-resolution(LR) image. The problem is of ill-posed nature as multiple solutions exist. The resolution of the constructed image depends on the upscaling factor and the resolution of the LR image. The constructed image should be of better quality than the normally magnified image. Thus not only does it retain the original details but also improves the image quality. The applications of image SR include medical imaging, remote sensing, robot vision, industrial inspection, satellite imaging, etc.

There are various techniques to approach this problem. We have opted to use a learning based SR by using a Convolutional Neural Network(CNN). The first stage includes training the model using patches of images from the training dataset. This sets the parameters/weights of the CNN model which are then further used to construct the HR image. Using this approach we double the image's size. To quadruple the size, we again feed the doubled output image in the same network. Hence, the same network can be reused for achieving further magnification factors. We further attempt to minimise the blur and blockiness in the image, by adding a regularization term of smoothness prior in the loss function, which optimises the output.

II. TOOLS AND TECHNOLOGIES USED

- Keras[5]
- Tensorflow[4]
- OpenCV
- Pickle
- Numpy
- Google Colab
- h5PY
- Python 2.7

III. CONVOLUTIONAL NEURAL NETWORKS[1]

Convolutional neural networks (CNNs) can be used in modeling a complex relationship between the input and the output data. They can be used in an unsupervised way to learn the features or in a supervised way for classification purpose. For this project, we use it to learn features in an unsupervised manner. The network architecture for this task consists of three kinds of layers namely convolutional, pooling and the fully-connected layer. Each layer, barring the pooling layer, is associated with the parameters or weights that correspond to a set of learnable filters. During the forward propagation, we convolve each filter across the input space, which results in a set of 2-dimensional (2D) activation maps corresponding to that filter. In other words, the network learns filters that activate when there is a specific feature at different spatial positions in the input. The filter weights are shared for the entire input image.

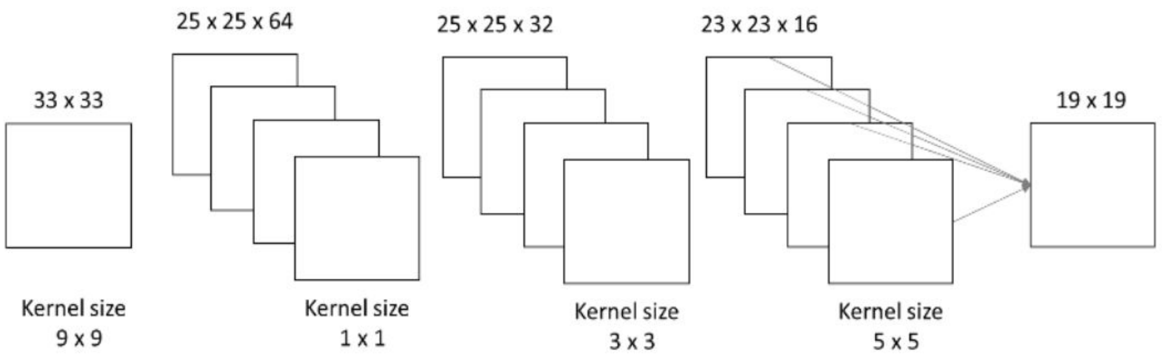


Fig.1 General block diagram of CNN used for super-resolution

Due to this weight-sharing property of CNN, there is a big reduction in the number of weights to be learned. Pooling layer is used for reducing the dimension and has a great significance while solving classification problem. In our work, since we are working on a reconstruction problem, we do not use the pooling layer. When a CNN has more than one hidden layer, then it is referred to as deep CNN (DCNN). These layers are added to learn hierarchical features between the input and the output.

A. Advantages of using CNN over other neural networks

• Sharing of weights i.e., reduced number of network parameters to be learned.
• Self-learning of filters by the network that eliminates the need for prior knowledge.
• Input data is 2D, preserving the structure and spatial dependencies.
• Less memory requirement since the same filter is applied to the entire input image.

B. Training a CNN

For learning the weights and parameters, one needs to train the CNN over a large amount of data, in our case images. The three steps involved in the training process of a CNN are:

B.1 Forward Propagation:

When we pass an image of size NxN, into the convolutional layer having filter size of m x m, denoted as ω, the output would result in a size of (N-m+1)x(N-m+1). To get the convolved output at location (i, j) of an l^{th} layer i.e., $x^l(i, j)$, we need to sum up the contributions from previous layer cells as shown in the equation below.

$$x^{\ell}(i,j)=\sum_{c=0}^{m-1}\sum_{d=0}^{m-1}\omega(c,d)y^{\ell-1}(i+c,j+d)$$

The equation above represents convolving the filter and the receptive field at (i, j). Here, $y^{l-1}(i+c, j+d)$ represents the convolved output of $(l-1)^{th}$ layer at location (i + c, j + d). Carrying out the convolution over the entire image results in a convolved image or convolution layer map. Use of a number of filters gives us a number of images in the convolution layer. A non-linearity applied on $x^l(i, j)$ for all i, j results in $y^l(i, j) = \sigma(x^l(i, j))$. σ represents a nonlinear activation function, i.e. ReLU(Rectified Linear Unit) in our case.

B.2 Backward Propagation for computing error derivatives with respect to weights (parameters):

Backpropagation occurs after each pass of forward propagation of every batch, where we try to reduce the value of the error function E, by taking its gradient with respect to the every weight ω(c,d) of the specific layer. Because of the weight sharing property, ω(c,d) occurs in the sum of all $x^l(i, j)$ expressions, and therefore backpropagation occurs throughout the intermediate hidden layers . The gradient is computed as shown below.

$$\frac{\partial E}{\partial\omega(c,d)}=\sum_{i=0}^{N-m}\sum_{j=0}^{N-m}\frac{\partial E}{\partial x^{\ell}(i,j)}\frac{\partial x^{\ell}(i,j)}{\partial\omega(c,d)}$$

$$=\sum_{i=0}^{N-m}\sum_{j=0}^{N-m}\frac{\partial E}{\partial x^{\ell}(i,j)}y^{\ell-1}(i+c,j+d)$$

B.3 Updating the weights:

The weights are updated by using the stochastic gradient descent (SGD) algorithm. For every example in the data, we compute the gradients with respect to weights using backpropagation, and then update the weights by the equation below. w and $x^{(i)}, y^{(i)}$ represent the weight vector and the ith training example. $\nabla_w E(w; x^{(i)}, y^{(i)})$ represents the gradient of the error function with respect to the network parameters, i.e. weights and α is the learning rate.

$$w := w - \alpha \nabla_w E(w; x^{(i)}, y^{(i)})$$

IV. WORK DONE IN BTP PART-1:

In order to get comfortable with coding CNN models in python we implemented a classification model on MNIST dataset which contains images of numbers ranging from 0-9. We have used tensorflow which is a neural network API. The MNIST dataset has 60000 images for training and 10000 images for testing. The training was done in batches of 100 images. The training includes approximately 20000 such steps. The CNN model is shown below.

Our CNN Structure on the TensorBoard tool:

Our built model follows the following CNN architecture[4]:

1. Convolutional Layer #1: Applies 32 5x5 filters (extracting 5x5-pixel subregions), with ReLU activation function.
2. Pooling Layer #1: Performs max pooling with a 2x2 filter and stride of 2 (which specifies that pooled regions do not overlap)
3. Convolutional Layer #2: Applies 64 5x5 filters, with ReLU activation function
4. Pooling Layer #2: Again, performs max pooling with a 2x2 filter and stride of 2
5. Dense Layer #1: 1,024 neurons, with dropout regularization rate of 0.4 (probability of 0.4 that any given element will be dropped during training to decrease overfitting).
6. Dense Layer #2 (Logits Layer): 10 neurons, one for each digit target class (0–9).

A. Results on Tensorflow's API:

We achieved an accuracy of 96.97% on the test data by training the model for 20000 steps, where 1 step indicates training the network on 1 batch(1 batch= 100 image vectors together). For improving the accuracy of the model at every step, we used gradient descent optimizer for backpropagation at a learning rate of 0.001, to optimize the filter values. We have also used a dropout rate of 0.4, to address the problem of overfitting.

Due to unavailability of validation tools in the Tensorflow API, we switched to Keras' API, to perform validation, on a little different model. For this we split the training dataset into two parts, one for training and the other for validation. The training is done in 12 epochs.

B. Results on Keras' API[5]:

Here we achieve an overall training and validation accuracy of about 99.15% after training the model on 12 epochs with a batch size of 128 images. The backpropagation optimization method used is Adadelta. The method dynamically adapts over time using only first order information and has minimal computational overhead beyond normal stochastic gradient descent.

V. BTP PART-2: PROPOSED APPROACH[1][2]

Coming back to Single Image Super-Resolution, we have used 91 images from the training dataset for model training. The model is trained for a scaling factor of 2[2]. Before feeding the image to the network, as preprocessing, we perform bicubic interpolation on the LR image to increase it's resolution. This bicubic-interpolated image is double the size of LR image as the scaling factor is 2. This new image, although double in resolution but is not of good quality. The images are given to the network in patches of 32 x 32. The network is trained to map this pre-processed patch to the HR patch. The network details are shown below:

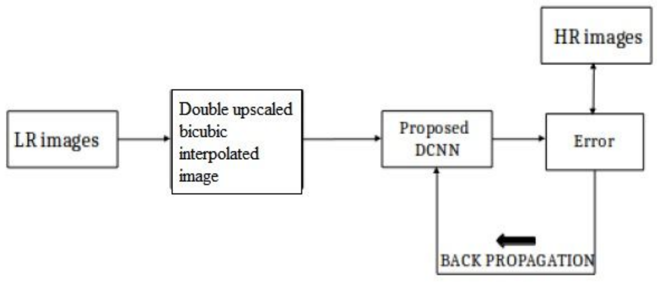


Fig. 2(a) Block Schematic of the training phase

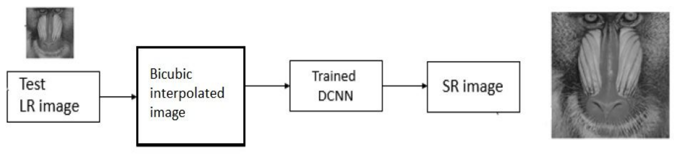


Fig. 2(b) Block Schematic of the testing phase

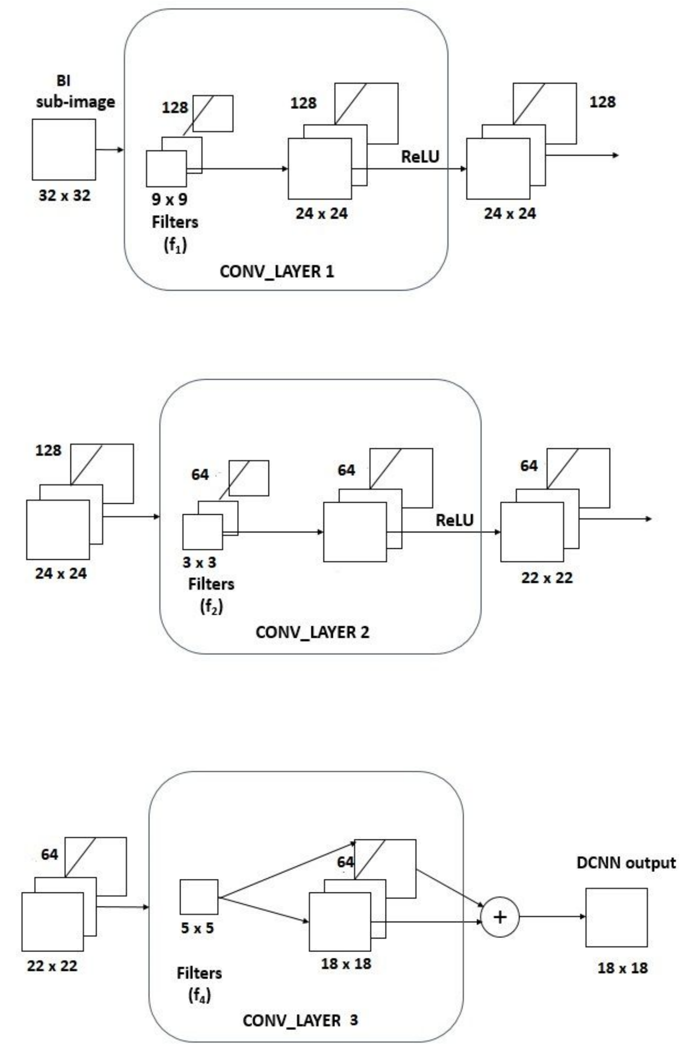


Fig. 3 Expanded view of the proposed DCNN model.

The reconstruction of these patches gives the complete HR image. After that, we added a regularization term in the loss function for smoothing effect in HR image.

VI. IMPLEMENTATION DETAILS:[1][6]

For performing training and testing experiments on the CNN, we have used the same dataset as in [6]. There are 91 color images of varying sizes. The 5 images in the Set5 section of the dataset is used to evaluate the performance for magnification factors of 2,4, and 8. As seen from Figure 3 above, our input consists of the sub-images/patches of size 32 x 32, and the size of our filters f1,f2,f3 are 9,3,5 respectively and their numbers/quantity of filters are 128, 64 and 1 respectively. We create our LR training images from all the 91 images by first downsampling them by removing alternate rows and columns, which undersamples them by a scale of 2. We then use bicubic interpolation to upscale it back to the original size and then create 32x32 patches/subimages out of it.

For training the DCNN, we prepare 32x32 pixel patches out from the bicubic-interpolated images and their corresponding HR images. These patches are extracted by using a stride of 11.To remove border effects, we do not use padding during training and hence our network results in a smaller output i.e. 18 x 18. Due to this, the mean squared error is calculated by computing the difference between central 18 x 18 crop of the ground truth (HR) sub-image of the corresponding LR input and the network output. The filter weights are initialized randomly using the samples from the Gaussian distribution with the zero mean and standard deviation of 0.001. With this training process, the network attempts to learn the non-linear mapping between the LR image patch and the corresponding HR image patch. We train the network over 50 epochs with a batch size of 128.

While testing the network, the same procedure goes, where the test image is converted to 32x32 patches, and is passed into the network. The network outputs the corresponding HR patch, which is then reconstructed in the LR image, in order to get the super-resolved(SR) image.

Further, to improve the smoothness of the image, we add a regularization parameter of a smoothness prior in the loss function of the neural network, which is as follows:

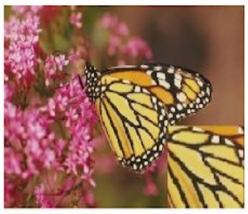
$$loss\ function = MSE(true\ image - reconstructed\ image)\ +$$

$$\lambda\Big(\sum_{i=1}^{m}\sum_{j=1}^{n}(reconstructed\ image(i,j) - reconstructed\ image(i,j-1))^2 +$$

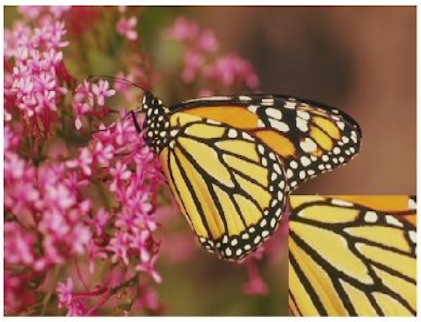
$$\sum_{i=1}^{m}\sum_{j=1}^{n}(reconstructed\ image(i,j) - reconstructed\ image(i-1,j))^2\Big)$$

where
MSE= Mean Square Error function
λ = Smoothness parameter
m,n = dimensions of the true image and the reconstructed image

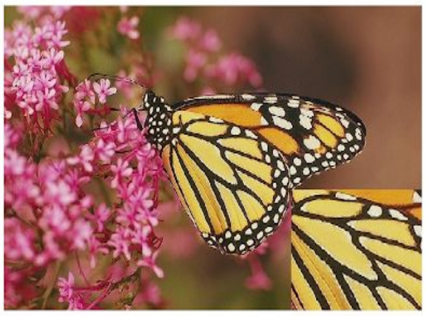
VII. EXPERIMENTAL RESULTS/ OBSERVATIONS:



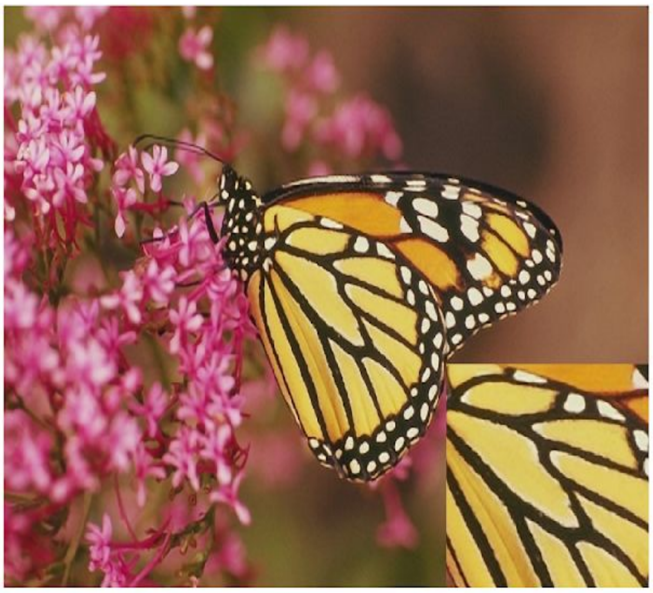
LR INPUT



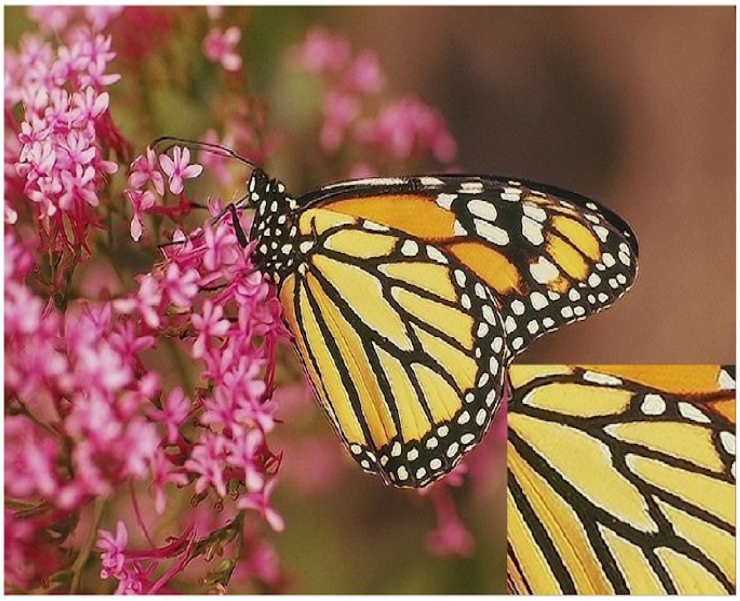
SR IMAGE WITH A MAGNIFICATION FACTOR(M.F.) OF 2



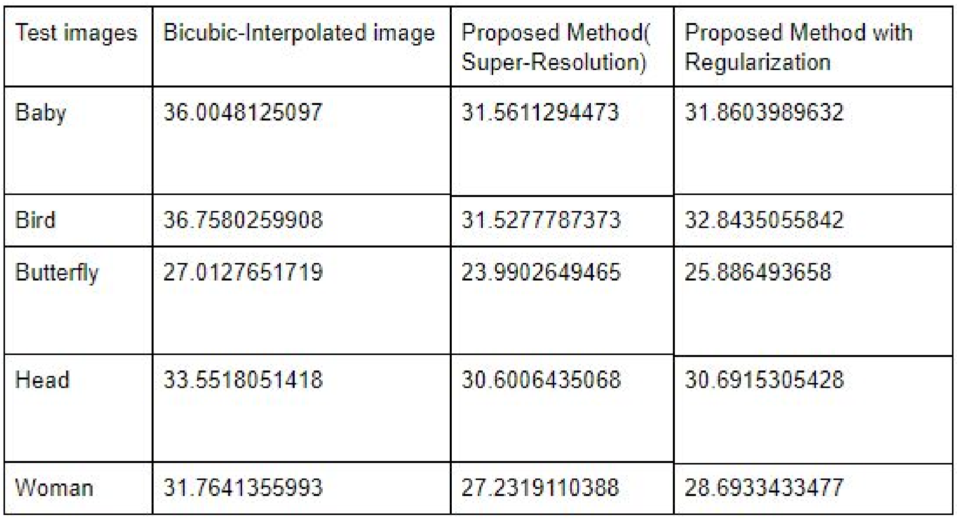
SR IMAGE WITH A M.F. OF 2 WITH SMOOTHING



SR IMAGE WITH A M.F. OF 4



SR IMAGE WITH A M.F. OF 4 WITH SMOOTHING



Test images	Bicubic-Interpolated image	Proposed Method(Super-Resolution)	Proposed Method with Regularization
Baby	36.0048125097	31.5611294473	31.8603989632
Bird	36.7580259908	31.5277787373	32.8435055842
Butterfly	27.0127651719	23.9902649465	25.886493658
Head	33.5518051418	30.6006435068	30.6915305428
Woman	31.7641355993	27.2319110388	28.6933433477

PSNR VALUES ON IMAGES OF TEST5 SET OF THE IMAGE DATABASE

The quality of the image is measured by computing PSNR(Peak Signal to Noise Ratio(PSNR) measured in db and the formula is as follows[1]:

$$PSNR = 10 * \log_{10}((255)^2/MSE)$$

where MSE is Mean Square Error between the ground truth image vector and the Super-resolved image vector.

VIII. CONCLUSION:

Through this project, we address the problem of super-resolution from computer-vision and machine-learning perspective. We propose a deep-learning based methodology involving Convolutional Neural Network, to perform super-resolution and produce finer details, which involves the learning of restoring details in the LR image. We attain higher magnification by reusing the neural network, and using the doubled super-resolved image as input, which in turn generates 4x magnification and similarly for 8x. We also tweak the loss function of the network with a regularization parameter to produce a more consistent output. The CNN filters and sub-images sizes play a crucial role in obtaining better results. The results obtained using this approach, show observable qualitative and quantitative improvements in the image quality, as compared to a digital zoom performed using bicubic interpolation.

Acknowledgment

Sincere thanks to Prof. Manjunath V. Joshi for his valuable guidance and suggestions for carrying out the experiments related to the approach. Also thanks to Prof. Andrew Ng’s coursera course of Machine Learning, which enabled us to seamlessly learn about the intricacies of Machine Learning and Neural Networks.

References

[1] Digital Heritage Reconstruction using deep learning-based Super-resolution by Prof. Manjunath V. Joshi and Prathmesh R. Madhu.

[2] Chao Dong, Chen Change Loy, Kaiming He, Xiaoou Tang. Learning a Deep Convolutional Network for Image Super-Resolution, in Proceedings of European Conference on Computer Vision (ECCV), 2014

[3] Super-Resolution via Deep Learning by Khizar Hayat Member, IEEE.

[4] Programming with Tensorflow python API learnt from https://www.tensorflow.org/tutorials/layers

[5] Programming with Keras python API learnt from https://keras.io/

[6] Timofte R, De Smet V, Van Gool L (2013) Anchored neighborhood regression for fast example-based super-resolution. In: The IEEE International Conference on Computer Vision (ICCV)