Image Denoising with CNNs

Aditya Rastogi, Ocima Kamboj

***Abstract*—The tremendous progress of convolutional neural networks (CNNs) in recent years have led to an increase in the use of CNNs for solving image denoising problem. Compared to classical denoising algorithms, CNNs have the advantage of fast testing and good performance. In this project, we implement a feed-forward convolutional neural network (DnCNNs) to handle different image restoration tasks. Apart from training the model for the removal of Additive White Gaussian Noise (AWGN) at a specific noise level, our model is also able to handle Gaussian denoising with unknown noise level (i.e. blind Gaussian Denoising). Residual learning and batch normalization are used to improve performance and speed up training. With the residual learning strategy, the DnCNN model removes the latent clean image from the noisy one in the hidden layers. This motivates the use of DnCNN for other image restroration tasks. A single model is trained to handle three general image denoising tasks - Gaussian Denoising, single image super-resolution, and JPEG Deblocking. We also show the effectiveness of the model in image deblurring task.**

***Index Terms*—Image Denoising, Convolutional Neural Networks, Single Image Super-resolution, JPEG Deblurring**

I. Introduction

During the past decade, convolutional neural networks have shown great success in handling various low-level vision tasks. Image denoising is one such long-standing problem in computer vision. The goal of image denoising is to recover the clean image x from the noisy image $y = x+v$. The assumption is that v is Additive White Gaussian Noise (AWGN). In general, image denoising methods can be grouped into two major categories - model based methods, and discriminative learning based. Model based methods like BM3D and WNNM are flexible in handling denoising problem with various noise levels, but their execution is time consuming, and they require the modelling of complicated priors. To overcome these drawbacks, discriminative methods have been recently developed.

The paper that we aim to implement is - Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising, by Kai Zhang et al. [1]. We refer to this as the base paper. The proposed denoising convolutional neural network is named DnCNN. Rather than directly outputing the clean image $\hat{x}$, the model is trained to predict the residual image $\hat{v}$, that is the difference between the noisy observation and the latent clean image. The batch normalization technique further improves and stabilizes the training performance of the DnCNN. Zhang et al. [1] showed that batch normalization and the residual learning strategy can mutually benefit from each other.

When v is the difference between the ground truth high resolution image and the bicubic upsmapled version of the low resolution image, the image degradation model can be converted to a single image super-resolution problem. Similarly, when v represents the difference between the original image and the compressed image, the JPEG image deblocking problem can be modeled by the same image degradation model. JPEG Deblocking refers to the removal of compression artifact that accompanies the application of lossy compression. In this sense, the SISR and JPEG deblocking can be considered as two special cases of a 'general' image denoising problem. We extend the DnCNN to handle these general image denoising tasks together, i.e. a single model is able to handle blind gaussian denoising, SISR, and JPEG Deblocking.

We successfully implement all the models from the base paper - namely DnCNN for handling a known level of gaussian noise, (named DnCNN-S), DnCNN for blind gaussian denoising (named DncNN-B), and DnCNN to handle three general image denoising tasks, namely Blind Gaussian Denoising, SISR, and JPEG Deblocking (named DnCNN-3).

In addition to implementation of the base paper, we conduct additional experiments on the denoising model. The base paper employs L2-loss as the loss function to train the network, also known as the mean squared error (MSE). Motivated by the fact that MSE is not an adequate metric to judge human visual perception, we employ the use of Structural SIMilarity (SSIM) index as a loss function [2]. To make a well-rounded observation, we also train the network with L1-loss, and with 'L1-loss + SSIM' as a combined loss function.

Taking inspiration from the base paper where they use the residual learning strategy to solve SISR and JPEG Deblocking problem with the same network, we extend the DnCNN model to another image restoration task - Image Deblurring. In this case, the residue v would be the difference between the clean image, and the image that has been degraded by a blurring operator.

The scope of this project is summarized as follows -

1) Successful implementation of the base paper in which an end-to-end deep CNN model is trained to handle guassian denoising with known level, blind gaussian denoising, and three general image denoising tasks - blind gaussian denoising, SISR and JPEG Deblocking.
2) Implementation of the denoising network with different loss functions. We show that L2-loss is not the best choice to train the network. This experimentation is not a part of the base paper.
3) We extend the model to handle the image restoration task of deblurring an image. This also is not a part of the base paper.
4) We have also implemented some state-of-the-art classical methods like BM3D, K-SVD etc. to compare our DnCNN model with.

The remainder of the report is organized as follows - in Section II. we provide a brief description of Residual Learning and Batch Normalization. In Section III, we present the architecture of the model. In Section IV, we go over the details and experiment results of the image denoising task - with known noise level, and with blind noise. The details and results of the general image denoising problem (Denoising+SISR+JPEG Deblocking) have been presented in Section V. In Section VI, we show the results of the image-deblurring problem.

II. Related Work

A. Residual Learning

Residual learning was proposed to ease the training of very deep networks [3]. It was observed that the training performance degrades as the depth of the network is increased. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a deep model leads to higher training error. In such a case, residual learning is easier to be learned. The intuition behind this is that if the input and the output is the same, that is the optimal function that has to be learned by the network is an identity mapping, then it is easier for the solver to drive the weights of the network to zero and learn the residue than learning the identity mapping.

B. Batch Normalization

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training as the parameters of the previous layers change. This phenomenon has been referred to as *internal covariate shift* [4]. Batch Normalization performs the normalization for each training mini-batch. This alleviates the problem of vanishing/exploding gradients and enable us to use larger learning rates.

C. Integration of Residual Learning and Batch Normalization

Zhang et al. [1] empirically find that the integration of residual learning and batch normalization speeds up and and stabilizes the training process, and also boosts the denoising performance. They show that -

- Residual learning benefits from batch normalization as batch normalization reduces the internal covariate shift. Even though residual learning without batch normalization has a fast convergence, it is still inferior to residual learning with batch normalization.
- Batch normalization benefits from residual learning. Without residual learning, batch normalization even has some adverse effect to training, and performs inferior to the one with both batch normalization and residual learning.

III. Network Architecture

The input to the DnCNN is the noise image $y = x + v$. The residual learning strategy aims to learn the mapping $\mathcal{R}(y) \approx v$, and then we have $x = y - \mathcal{R}(y)$. The mean squared error between the desired residual images and the estimated ones can be used as the loss function to train the network. Figure-1 shows the architecture of our DnCNN model. If D is the depth of the network, then there are three types of layers -

1) Conv+ReLU - for the first layer. Let c represent the number of colour channels. For grayscale images, $c = 1$, and for coloured images $c = 3$. 64 filter of size $3 \times 3 \times c$ are used to generate 64 feature maps, and ReLU activation function is used for non-linearity.
2) Conv+BN+ReLU - for layer-2 to layer-(D-1). 64 filters of size $3 \times 3 \times 64$ are used, and batch normalization [4] is added between the convolution layer and the ReLU layer.
3) Conv- for the last layer. c filters of size $3 \times 3 \times 64$ are used to reconstruct the output.

To train the network for denoising Gaussian noise of known noise level, the depth D of the network is set to 17. For training the network for blind denoising and the general image denoising task, we set the network depth to 20. For the deblurring task, the network, depth is set to 25.

IV. Gaussian Denoising

A. Experimental Setting

For all the denoising networks, we use the Berkeley Segmentation Dataset for training and testing [7]. The BSDS500 dataset is a data set of 500 natural images. A standard subset of the BSDS500 dataset comprising of 68 images, known as BSD68, is used for testing purposes. Out of the remaining 432 images, 400 are used for training and 32 are used for validation.

To train DnCNN for Gaussian denoising with known noise level, we cut $128 \times 2000 = 256,000$ patches of size 40×40 from the training set of 400 images, and 20,480 patches of size 40×40 from the validation set of 32 images. We consider three noise levels - $\sigma =$15, 25 and 50. The training and validation data set was created in MATLAB using the 'imnoise' function. The data set is augmented by performing rotation and flip operations on the patches. Augmentation helps in improving generalization. Before feeding the inputs to the network, they are converted from the standard 'uint8' format of images to 'single' format. The data augmentation step, and the conversion of format from 'uint8' to 'single' is common across all our models, so we won't be describing it again with each model. The depth of the network is set to 17. The ADAM optimizer is used for all the models with different learning rates and different mini-batch size as required. In this case, we used learning rate of 0.0002 and mini-batch size of 128 for noise level 25 and 50. For the case of noise level 15, the learning rate was kept 0.0002 and a mini batch size of 500 was used. This model is referred to as *DnCNN-S*.

To train a single DnCNN model for handling blind gaussian denoising, we set the range of noise levels as $\sigma \in [0, 55]$. We cut $128 \times 3000 = 384,000$ patches of size 50×50 from the training set of 400 images, and 30,720 patches of size 50×50 from the validation set of 32 images. The patch size is increased so as to get more contextual information required

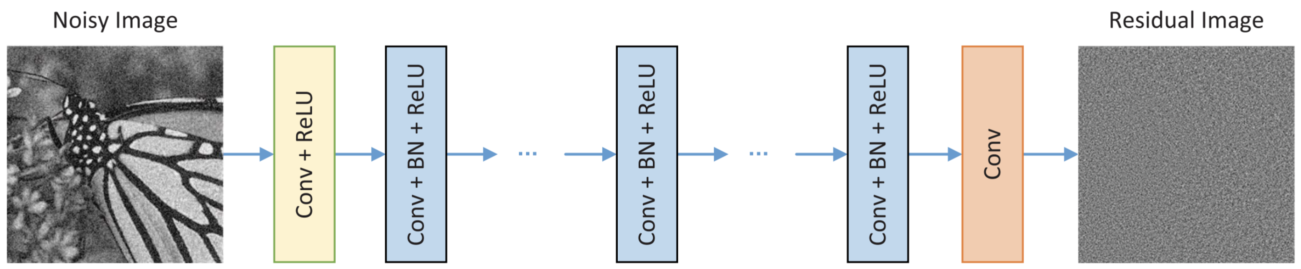


Fig. 1: Network Architecture (figure has been taken from [1])

for the blind denoising task. While creating the data set for training and validation, the noise level to be added, σ, is generated randomly from the specified range of $\sigma \in [0, 55]$. The depth of network is set to 20. We use learning rate of 0.0001 and mini-batch size of 100. We refer to the single DnCNN model for blind gaussian denoising task as *DnCNN-B*

The coloured version of DnCNN-B is named CDnCNN-B. The training and validation data are created in exactly the same way. We used a learning rate of 0.0002 and mini-batch size of 100. The network depth is set to 20.

For all the above described models, the mean-square error (MSE) is adopted as the loss function.

It is well known that MSE is not an adequate metric to quantify human visual perception [2]. This motivated us to see the effects of changing the loss function to SSIM for training purposes. More accurately, we use (1-SSIM) as the loss function, because this is the function that needs to be reduced. We refer to (1-SSIM) as DSSIM. The same training and testing data sets and network depth as DnCNN-S and DnCNN-B are used for known gaussian denoising and blind gaussian denoising respectively. We name the corresponding models with DSSIM loss function as *SSIMDnCNN-S* and *SSIMDnCNN-B*. For SSIMDnCNN-S, we used a learning rate of 0.005 and minibatch size of 128 for $\sigma = 15$, and mini-batch size of 100 for $\sigma = 25$ and 50.

To be able to make a more well rounded observation on the use of different loss functions, we trained the blind gaussian denoiser with L1-loss as the loss function. This model is referred to as *L1DnCNN-B*. The learning rate used was 0.005 and mini-batch size was 50.

After observing that the model trained with L1 loss performs consistently better than the one trained with L2-loss, we trained another model for blind gaussian denoising with a composite loss function of L1 and DSSIM. The composite loss function is framed as a weighted average - α L1 + $(1-\alpha)$DSSIM. After a few experiments, and motivated by the fact that L1 loss was giving better results for some metrics as compared to DSSIM, we set the value of α as 0.8. A learning rate of 0.004 and mini-batch size of 400 was used. This model is referred to as *MixDnCNN-B*.

We initialize the weights of the network according to a random normal distribution with mean 0 and standard deviation 0.001. We train the model for 50 epochs.

The training and testing codes were written in Keras. All the pre-processing and post-processing was done in MATLAB. The training was carried out using SERC's Nvidia-DGX cluster. All the models with known noise level took about 3 hours for training. The models for blind gaussian denoising took around 8 hours to train.

B. Compared Methods

We compare our DnCNN model with several state-of-the-art denoising methods.

1) BM3D: Block Matching Block Matching 3D [5] is a state of the art approach for recovering images corrupted by AWGN (additional white gaussian noise). BM3D is based on the statistic that in clean natural images there are non local similarities of each small patch. These are termed as non local structural similarities. The algorithm matches each small patch of the image with similar patches in the image and stacks them together in a box. As the images patches that are stacked on top of each other are very similar in structure, it allows us to achieve a sparse representation of the image in a transformed domain (DCT, DFT etc). The algorithm proceeds in two phases. In phase one, it hard threshold's the coefficients of the 3D transformation T_{3D} of the block to filter out the noise, the filtered patches are places at their original place in the image. In second phase it again forms the blocks from the output of phase one and filters it using Weiner filter in T_{3D} domain and again places the patches in their original position. A problem with this algorithm is that it requires an estimate of the variance of noise σ_z^2

2) Weiner Filter: Weiner filtering is a widely used classical algorithm for recovering image from noisy image corrupted by AWGN. The biggest problem with using a simple low pass filter on an image is that we lose the high pass components of the image. This algorithm assumes the statistics of higher frequency component of the clean image to be Normally distributed and using that computes an estimate of the high pass component of the clean image. The estimate of high pass clean image is given as.

$$\hat{X}_1 = \frac{\sigma_{x_1}^2}{\sigma_{x_1}^2 + \sigma_{z_1}^2}$$

TABLE I: The average PSNR(DB) results of different methods on the BSD68 dataset. Best results are highlighted in bold

Methods	Noisy	Weiner Filter	KSVD	BM3D	DnCNN-S	DnCNN-B	SSIMDnCNN-S	SSIMDnCNN-B	L1DnCNN-B	MixDnCNN-B
σ =15	24.37	28.66	28.91	30.78	**31.20**	31.13	31.06	30.96	31.16	31.12
σ =25	19.97	25.70	27.46	28.16	**28.79**	28.67	28.28	28.38	28.72	28.66
σ =50	14.81	21.10	23.52	24.60	25.77	25.82	25.49	25.43	**25.94**	25.86

TABLE II: The average SSIM results of different methods on the BSD68 dataset. Best results are highlighted in bold

Methods	Noisy	Weiner Filter	KSVD	BM3D	DnCNN-S	DnCNN-B	SSIMDnCNN-S	SSIMDnCNN-B	L1DnCNN-B	MixDnCNN-B
σ =15	0.5535	0.7771	0.8516	0.8727	**0.8905**	0.8831	0.8859	0.8851	0.8824	0.8851
σ =25	0.3705	0.6413	0.7672	0.7973	0.8178	0.8150	**0.8239**	0.8194	0.8186	0.8209
σ =50	0.1930	0.4109	0.6094	0.6657	0.7035	0.7056	0.7160	0.7129	0.7110	**0.7177**

where $\sigma^2_{x_1}$ and $\sigma^2_{z_1}$ are the variance of high pass image X_1 and noise estimate Z_1.
The algorithm we implemented is called Local Adaptive Weiner filtering where the above mentioned variances are calculated in small patch sizes. The code for Adaptive Weiner filter is attached in appendix. A problem with this algorithm is that it requires an estimate of the variance of noise σ^2_z

3) K-SVD: Given the input noisy image Y , the goal of the K-SVD [6] algorithm is to learn a dictionary D such that the estimated clean image $\hat{X}$ can be reconstructed as a sparse combination of D, i.e we want to minimize $||D\alpha - Y||^2_2$ (where $D\alpha = \hat{X}$) such that most of the values of α are zero. The dictionary is learned by taking the DCT coefficients of small patches of the image and converting them into a vector. The pseudocode of the algorithm is provided in the paper. The algorithm is implemented in two steps, first we find the α that minimizes the above given cost function. After that we update the dictionary by the rule given in paper. For implementing the first step we have used Orthogonal Matching Pursuit algorithm and the code for this particular solver is taken from third party. Similar to the methods mentioned above, this method suffers from the problem that for good results, some idea of the input noise variance is required. The implementation is given in the appendix.

C. Quantitative and Qualitative Evaluation

The average PSNR results of different methods are shown in Table-I, and the average SSIM results are shown in Table-II. Our DnCNN model outperforms the benchmark BM3D for all noise levels. In fact, there is an appreciable improvement in SSIM for DnCNNs over BM3D.

For PSNR metric in Table-I, DnCNN-S gives the best results for two out of three values of σ, which is expected because DnCNN-S has been trained for a particular noise level. It is interesting to note that for high noise level of $\sigma = 50$, the blind denoising model DnCNN-B performs better than DnCNN-S, and the best performance is of L1DnCNN-B. In fact, the blind denoising model trained with L1-loss, namely L1DnCNN-B, performs better than the blind denoising model trained with L2-loss, namely DnCNN-B, for all the three noise levels, which is surprising as one would expect the model trained with L2-loss function to generate better PSNR values.

TABLE III: The average value of different quality metrics for different cost functions on test set with σ =15 for blind denoising model. Best results are highlighted in bold.

Blind Denoising σ =15	Training Cost Function			
Image Quality Metric	L2	DSSIM	L1	Mix
RMSE	7.340	7.506	**7.336**	7.372
L1	5.303	5.364	5.248	**5.244**
PSNR	31.13	30.96	**31.16**	31.12
SSIM	0.8831	0.8851	0.8824	**0.8851**

TABLE IV: The average value of different quality metrics for different cost functions on test set with σ =25 for blind denoising model. Best results are highlighted in bold.

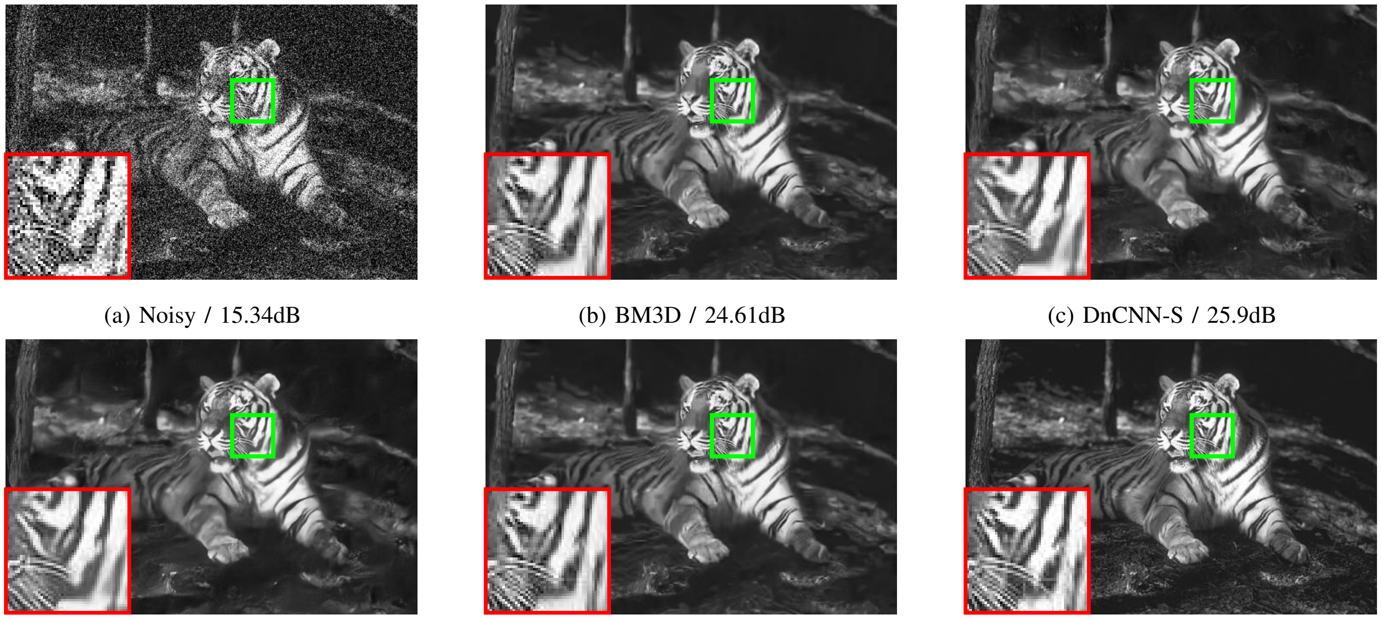
Blind Denoising σ =25	Training Cost Function			
Image Quality Metric	L2	DSSIM	L1	Mix
RMSE	9.820	10.246	**9.781**	9.857
L1	6.969	7.341	**6.883**	6.916
PSNR	28.67	28.38	**28.72**	28.66
SSIM	0.8150	0.8194	0.8186	**0.8209**

TABLE V: The average value of different quality metrics for different cost functions on test set with σ =50 for blind denoising model. Best results are highlighted in bold.

Blind Denoising σ =50	Training Cost Function			
Image Quality Metric	L2	DSSIM	L1	Mix
RMSE	13.691	14.306	**13.486**	13.642
L1	9.626	9.959	**9.397**	9.484
PSNR	25.77	25.43	**25.94**	25.86
SSIM	0.7035	0.7129	0.7110	**0.7177**

For SSIM metric in Table-II, different models perform the best for the three different noise levels. One would expect that the model trained with the DSSIM loss function would generate the best values of SSIM on the test set, which is not the case. It is also worth noticing that DnCNN-B is outperformed by MixDnCNN-B for all the three noise levels.

For making an effective comparison of the performance of the networks with different loss functions, we show the values of four different quality metrics for the blind denoising network with the four loss functions - MSE, DSSIM, L1, and Mix(L1+DSSIM). Table-III contains the result for $\sigma = 15$,



(a) Noisy / 15.34dB (b) BM3D / 24.61dB (c) DnCNN-S / 25.9dB

(d) DnCNN-B / 25.84dB (e) L1DnCNN-B / 25.85dB (f) Ground Truth

Fig. 2: Denoising results of one image from BSD68 with noise level 50

TABLE VI: The average value of different quality metrics for CDnCNN-B. Best results are highlighted in bold.

Noise Level	σ=15		
METRIC	Noisy	BM3D	CDnCNN-B
PSNR	27.097	30.862	**31.925**
SSIM	0.831	0.938	**0.949**
Noise Level	σ=25		
PSNR	22.762	28.569	**30.055**
SSIM	0.670	0.899	**0.925**
Noise Level	σ=50		
PSNR	17.467	25.542	**27.319**
SSIM	0.426	0.832	**0.872**

Table-IV contains the result for $\sigma = 25$, Table-V contains the result for $\sigma = 50$. For most of the cases, the network trained with L1-loss provides the best results on all the metrics (except SSIM). For the remaining ones, the network trained with the composite L1+DSSIM loss performs the best. The mixed loss functions consistently provides the best results for SSIM metric.

We would like to point out that DnCNN-B doesn't give the best quality metric in any case, which shows that L2-loss isn't the best loss function to train the network. L1 loss performs better than L2 loss, because L1-loss does not over-penalize large errors.

For color image denoising, we have compared our CDnCNN-B model with the benchmark BM3D in Table-VI. The PSNR gain for CDnCNN-3 over BM3D is around 1 DB for $\sigma = 15$, 1.5 DB for $\sigma = 25$ and almost 2 DB for $\sigma = 50$, which is appreciable. The SSIM improvement of CDnCNN-3 over benchmark BM3D is around 0.01 for $\sigma = 15$, 0.026 for $\sigma = 25$ and 0.04 for $\sigma = 50$. The results of CDnCNN-3 get progressively better over BM3D for higher noise levels.

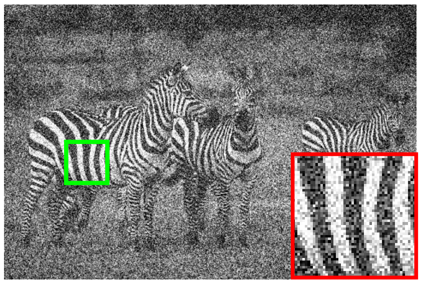
Figure-2, 3 and 4 illustrate the visual results of different methods. We can see that even with noise level as high as 50, our models yield visually pleasant results. Figure-4 shows that our CDnCNN-B model is better able to retain sharp edges and fine details as compared to BM3D.

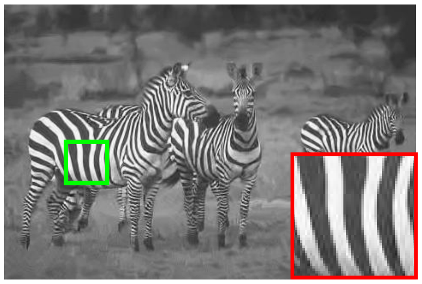
D. *Some Additional Remarks*

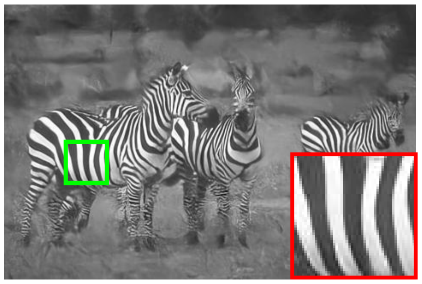
The PSNR values that we achieved on our test set were about 0.5 db lower as compared to the ones reported in the base paper for all the methods - namely, DnCNN-S, DnCNN-B as well as BM3D. When we investigated the issue, we realized that the issue was arising because of a difference in the way we created our training data. In the AWGN literature, there exist two widely used settings - unclipping and clipping [8]. In the unclipping method, the noisy image is not clipped in the range of 0-255, or more precisely quantized into 8-bit format, after the addition of Gaussian Noise. This serves as an ideal case for testing different denoising methods. In the clipping method, the Noisy image is quantized into the 8-bit format. This is what we followed, which degraded our results by a little margin as the clipping of noisy input leads to a deviation of the noise characteristics from the ideal AWGN. The clipping method seemed more intuitive to us as real images would always be in the 8-bit format, and that is why we followed this procedure in all our models.

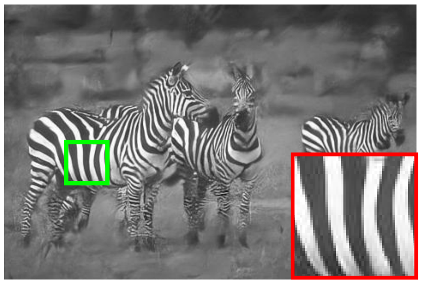
V. General Image Denoising - Gaussian Denoising, SISR, JPEG Deblocking

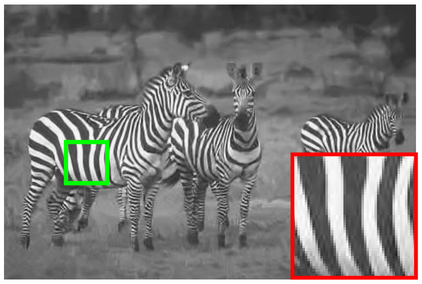
When the residual represents the difference between the latent clean image and the degraded observation, then DnCNN


(a) Noisy / 14.42dB


(b) BM3D / 23.65dB


(c) DnCNN-S / 24.95dB


(d) DnCNN-B / 24.85dB


(e) L1DnCNN-B / 24.91dB

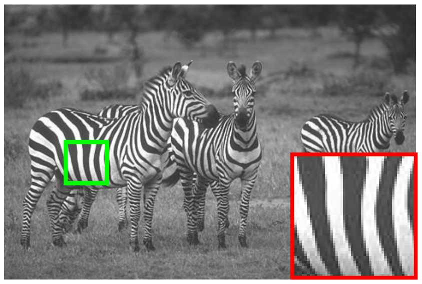
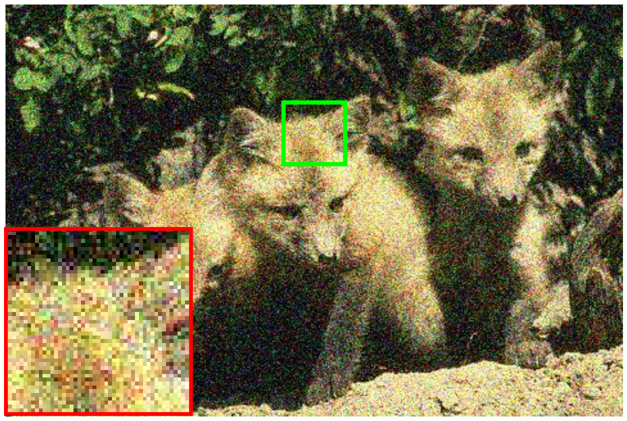
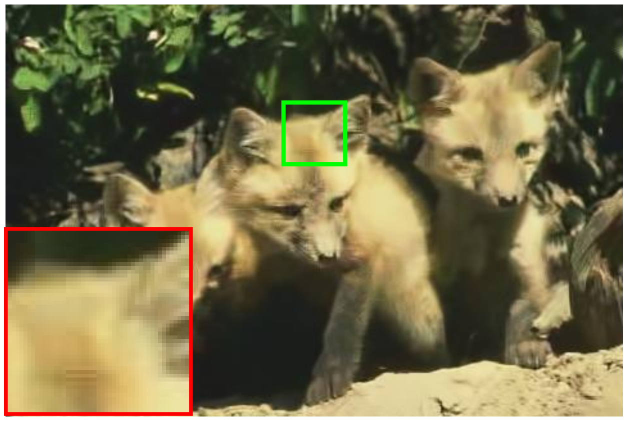
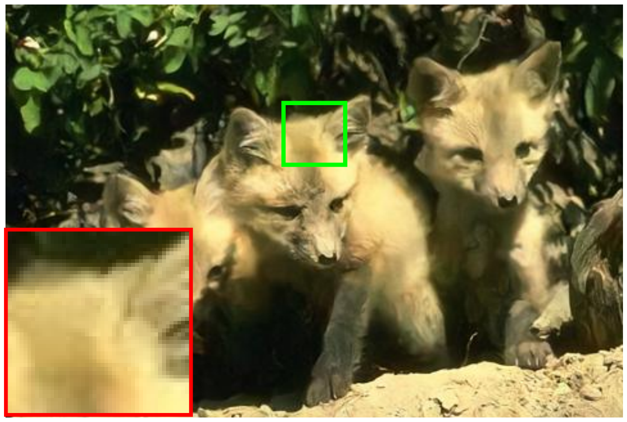

(f) Ground Truth

Fig. 3: Denoising results of one image from BSD68 with noise level 50


(a) Noisy / 17.92dB


(b) BM3D / 24.06dB


(c) CDnCNN-B / 27.05dB

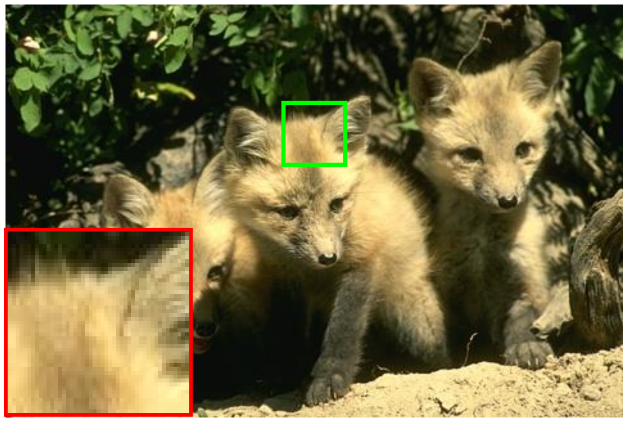

(d) Ground Truth

Fig. 4: Denoising results of one image from BSD68 with noise level 50

can be adapted to handle a more general image restoration task. In this section, we extend our DnCNN model to handle three general image denoising tasks - gaussian denoising, single image super resolutoin, and JPEG image deblocking.

A. Experimental Setting

We use the same BSDS500 data set for training and validation. Specifically, 400 images are used for training the network, and 32 are used for validation.

All the data preprocessing was done in MATLAB. The noisy image for Gaussian denoising input is generated by adding gaussian noise to the clean image with the value of σ generated randomly between $[0, 55]$. This was done using MATLAB's 'imnoise' function. The SISR input is created by first bicubic downsampling and then bicub upsampling the high-resolution image with scaling factors of 2, 3 and 4. These scale factors are generated randomly. The bicubic upsampling and downsampling is carried out using MATLAB's 'imresize' function. The JPEG Deblocking input is generated by compressing the image with a quality factor ranging from 5 to 99 using MATLAB's 'imsave' function. The quality factors are generated randomly. All these images are treated as inputs to a single DnCNN model. Any given input patch is degraded randomly according to one of these three image degradations. In total, we generate $128 \times 8000 = 1,024,000$ patches of size 60×60 for training from the training set of 400 images, and 81,920 patches of size 60×60 were created for validation from the validation set of 32 images. Rotation/flip based operations are used to augment the data. As before, the inputs are converted from the 'uint8' format to 'single' format before being fed to the network. The depth of network is set to 20. ADAM optimizer is used with a learning rate of 0.0002 and a mini-batch size of 128. We refer to this model as *DnCNN-3*.

The coloured version of DnCNN-3 is named CDnCNN-3. The training and validation set are created in exactly the same way. ADAM optimizer with a learning rate of 0.0002 and mini-batch size of 100 was used for training.

L2-loss is adopted as the loss function. For DnCNN-3 and CDnCNN-3, the weights are initialized with the learned parameters of DnCNN-B and CDnCNN-3 respectively. The model is trained for 50 epochs. The training and testing codes were written in Keras. All the pre-processing and post-processing was done in MATLAB. The training was carried out using SERC's Nvidia-DGX cluster. DnCNN-3 took about 26 hours to train, and CDnCNN-3 took about 30 hours to train.

For training purposes, different test data sets are used for each task out of Gaussian Denoising, SISR, JPEG Deblocking -

- For Gaussian Denoising, BSD68 is used for testing the performace.
- For SISR, 3 testing datasets are adopted - Set5, Set 14 and Urban100.
- For JPEG Deblocking, Classic5 and LIVE1 are adopted as testing data sets.

These data sets are in accordance with the ones use in the base paper.

B. Quantitative and Qualitative Evaluation

TABLE VII: Average PSNR(DB) and SSIM values for DnCNN-3

Gaussian Denoising					
Dataset	Noise Level	Noisy Input		DnCNN-3	
		PSNR	SSIM	PSNR	SSIM
BSD68	15	24.37	0.5535	31.14	0.8816
	25	19.97	0.3705	28.73	0.8162
	50	14.81	0.1930	25.95	0.7087

Single Image Super-resolution					
Dataset	Upscaling Factor	Bicubic upsampled Input		DnCNN-3	
		PSNR	SSIM	PSNR	SSIM
Set5	2	32.33	0.9205	35.27	0.9493
	3	27.31	0.8235	28.33	0.8509
	4	27.21	0.7946	29.81	0.8645
Set14	2	28.33	0.8509	29.98	0.8878
	3	24.78	0.7256	24.93	0.7514
	4	24.11	0.6729	25.13	0.7198
Urban100	2	25.35	0.8196	28.35	0.8822
	3	21.38	0.6584	21.02	0.6752
	4	21.78	0.6363	23.59	0.7202

JPEG Image Deblocking					
Dataset	Quality Factor	Compressed Input		DnCNN-3	
		PSNR	SSIM	PSNR	SSIM
Classic5	10	27.97	0.7834	29.47	0.8241
	20	30.57	0.8636	31.74	0.8826
	30	32.00	0.8951	32.86	0.9055
	40	32.74	0.9079	33.41	0.9157
LIVE1	10	26.98	0.7783	28.19	0.8111
	20	29.39	0.8615	30.35	0.8782
	30	30.71	0.8939	31.45	0.9043
	40	31.37	0.9066	31.99	0.9151

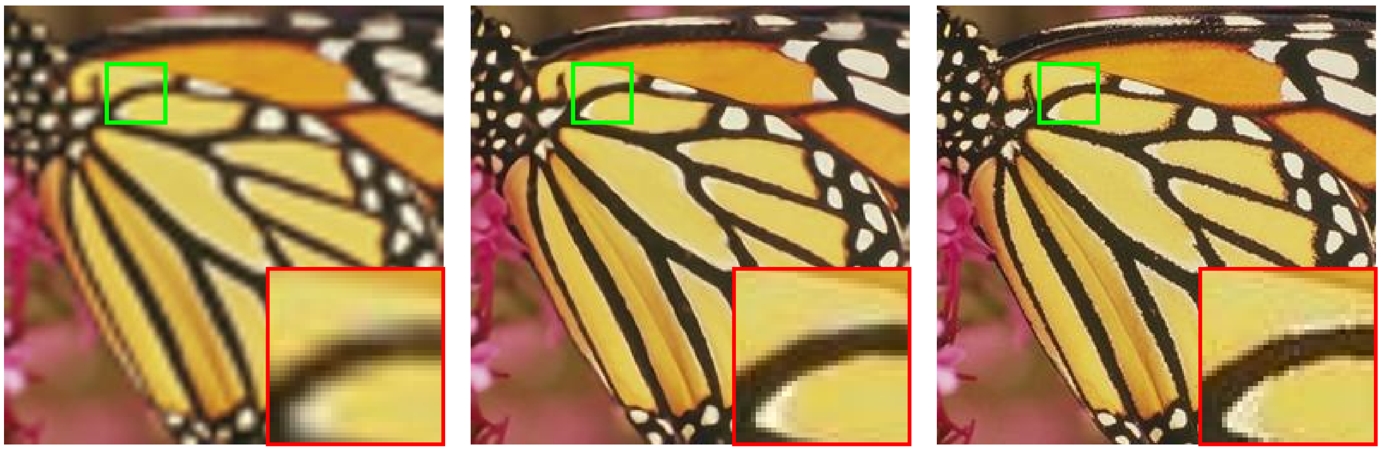
TABLE VIII: Average PSNR(DB) and SSIM values for CDnCNN-3

Gaussian Denoising					
Dataset	Noise Level	Noisy Input		DnCNN-3	
		PSNR	SSIM	PSNR	SSIM
BSD68	15	27.10	0.8314	32.09	0.9512
	25	22.76	0.6695	30.20	0.9271
	50	17.47	0.4258	27.47	0.8743

Single Image Super-resolution					
Dataset	Upscaling Factor	Bicubic upsampled Input		DnCNN-3	
		PSNR	SSIM	PSNR	SSIM
Set5	2	32.16	0.9693	34.58	0.9801
	3	27.13	0.9211	28.00	0.9304
	4	27.06	0.9138	29.38	0.9462
Set14	2	28.41	0.9109	29.85	0.9321
	3	24.80	0.8325	24.95	0.8449
	4	24.14	0.8011	25.04	0.8255
Urban100	2	25.35	0.8196	28.35	0.8822
	3	21.35	0.7532	21.04	0.7630
	4	21.77	0.7410	23.49	0.8015

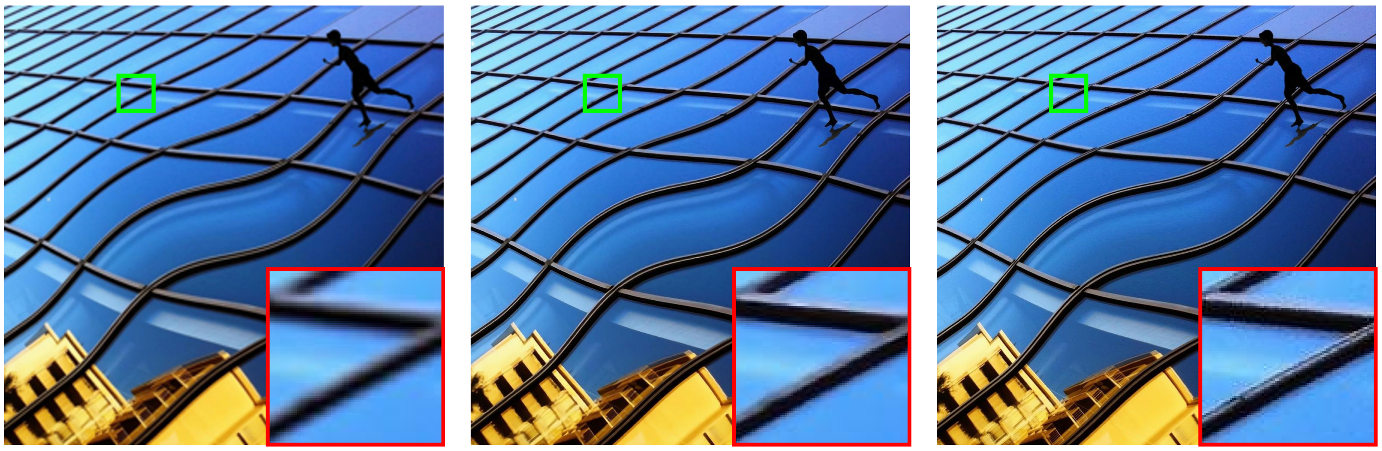
JPEG Image Deblocking					
Dataset	Quality Factor	Compressed Input		DnCNN-3	
		PSNR	SSIM	PSNR	SSIM
LIVE1	10	25.87	0.8219	27.12	0.8535
	20	28.49	0.8982	29.44	0.9128
	30	29.88	0.9244	30.59	0.9325
	40	30.59	0.9347	31.18	0.9414

Table-VII and Table-VIII list the average PSNR and SSIM results of DnCNN-3 and CDnCNN-3 respectively for different



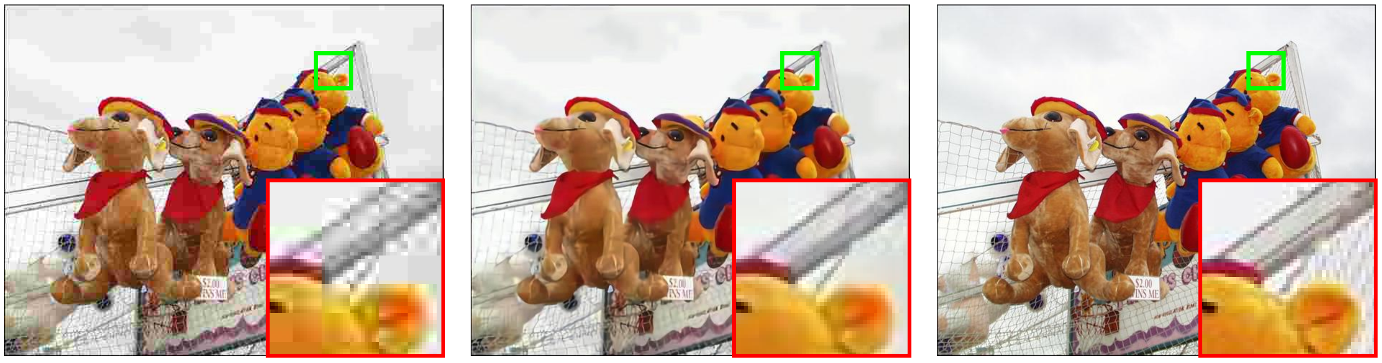
(a) Bicubic Upsampled Version (b) CDnCNN-3 (c) Ground Truth

Fig. 5: Single image super-resolution of 'monarch' from Set5 dataset with scale factor of 4



(a) Bicubic Upsampled Version (b) CDnCNN-3 (c) Ground Truth

Fig. 6: Single image super-resolution of one from Urban100 dataset with scale factor of 4



(a) Compressed Image (b) CDnCNN-3 (c) Ground Truth

Fig. 7: JPEG image deblocking results of "Carnivaldolls" from LIVE1 dataset with quality factor 10

general image denoising tasks. Figures - 5, 6, 7, 8 illustrate the visual results of our model. Specifically, figure-9 and figure-10 is a good example of the capability of our network to handle a general image denoising task. In these aforementioned figures, different patches of the image have been degraded with different degradation model. Even then CDnCNN-3 produces

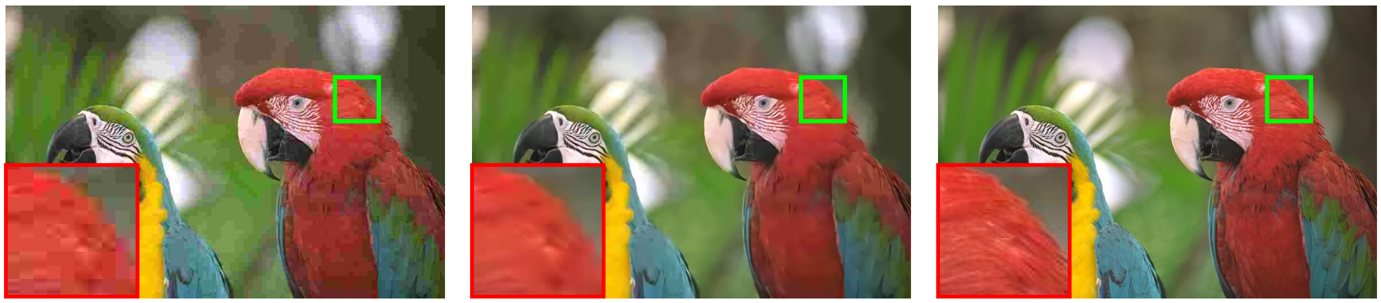

(a) Compressed Image (b) CDnCNN-3 (c) Ground Truth

Fig. 8: JPEG image deblocking results of ”Parrots” from LIVE1 dataset with quality factor 10

a natural looking image.

VI. Image Deblurring

Motivated by the success of the DnCNN model in handling a general image denoising task, we extend the model to handle another image restoration task - Image Deblurring.

A. *Experimental Setting*

Like before, the BSDS500 data set is used for creating training, validation, and testing data with 400, 32 and 68 images respectively.

The data preprocessing was done in MATLAB. MATLAB's 'fspecial' function was used to create a motion blur kernel. The length of the motion blur was selected randomly from [5,15], and the angle was generated randomly from $[0°, 360°]$. In Total we generate $250,000$ training patches of size 100×100 from the training set of 400 images, and $20,000$ validation patches of size 100×100 are generated from the validation set of 32 images. The patch size was finalized after conducting a few experiments. We initially started from a patch size of 50×50, but found that the network could not converge. A bigger patch size allows the network to get more contextual information, which is essential for the deblurring task. Rotation/flip based operations are used to augment the training set. As before, the inputs are converted from the 'uint8' format to 'single' format before being fed to the network.

The depth of network is set to 25. The reason for increasing the depth of the layer is that our initial experiments for deblurring were done using a motion blur kernel with a known level of length of the blur. The network depth was set to 20 for handling this task. Even with such a network, the model had difficulty converging. When we finally able to get good results for a patch size of 100, we followed the same intuition that was applied when we went form DnCNN-S to DnCNN-B, which is, to handle the blind denoising problem, increase the depth of the network. That is why we set the number of layers to 25.

ADAM optimizer is used with a learning rate of 0.0002 and a mini-batch size of 50. We refer to this model as *CDnCNN-Deblur*.

L2-loss is adopted as the loss function. The weights of the network are initialized according to a random normal distribution with zero mean and a standard deviation of 0.001. The model is trained for 80 epochs. The training and testing codes were written in Keras. All the pre-processing and post-processing was done in MATLAB. The training was carried out using SERC's Nvidia-DGX cluster. CDnCNN-Deblur took about 41 hours to train.

For testing purpose, we use the BSD68 dataset.

B. *Quantitative and Qualitative Evaluation*

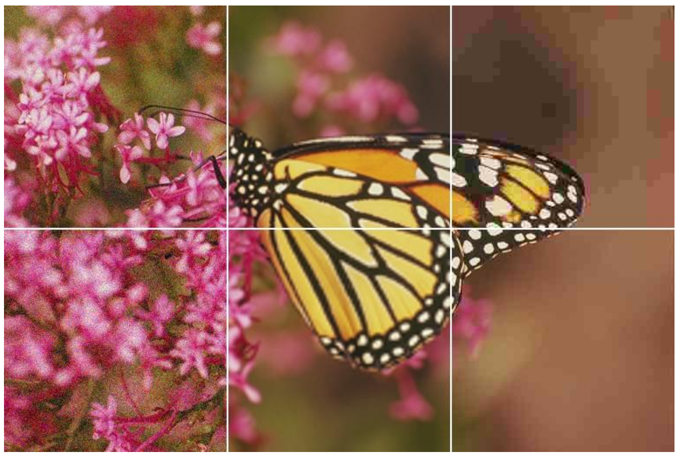
Table-IX lists the average PSNR and SSIM results of CDnCNN-Deblur. Figures - 11, 12, and 13 illustrate the visual results of our model. The CDnCNN-Deblur model is successfully able to deblur the image. It also retains the sharp edges to a huge margin.

TABLE IX: The average value of PSNR(dB) and SSIM for CDnCNN-Deblur.

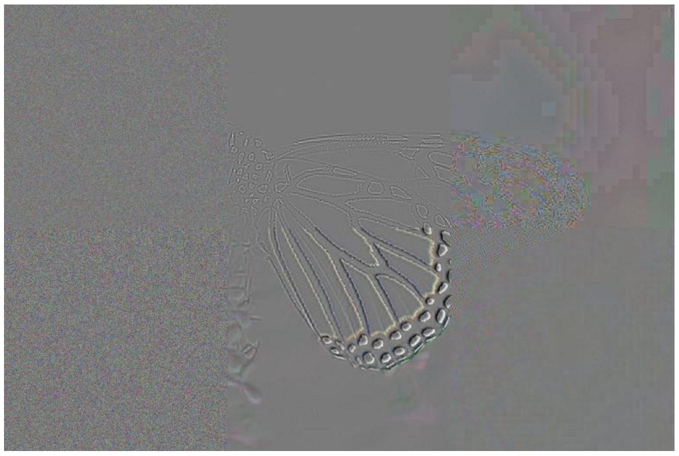
	PSNR (DB)	SSIM
Noisy Image	23.93	0.797
CDnCNN-Deblur	28.99	0.924

VII. Conclusion

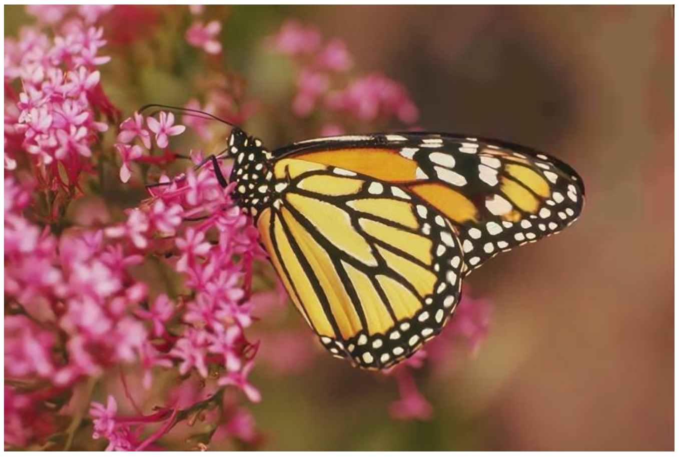
In this project, we successfully implemented our base paper for image denoising task. A deep CNN was implemented for image denoising. Residual Learning and Batch Normalization were used to speed up training and boost the denoising performance. Unlike traditional discriminative models which train specific models for certain noise levels, the DnCNN model can also handle blind gaussian denoising. We go a step further from the base paper and train the denoising network with different loss functions. We show that L2 isn't the best loss function to train the network, rather L1-loss gives us better results for all the metrics than L2-loss. We also implemented the general image denoising network from the base paper successfully. This network is able to handle three general image denoising tasks - gaussian denoising, SISR with multiple scaling factors, and JPEG Deblocking with different quality factors. We extended the sample principle of residual learning to train the network for image Deblurring, which was not a part of the base paper. This model is successfully able to handle the deblurring task.



(a) Input Image

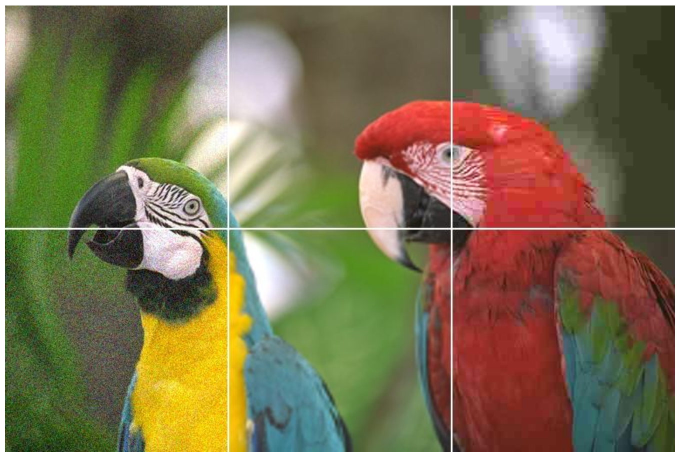


(b) Output Residual Image

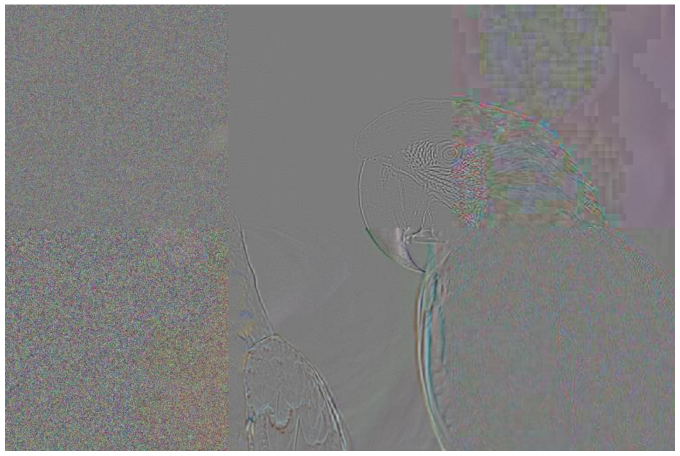


(c) CDnCNN-3

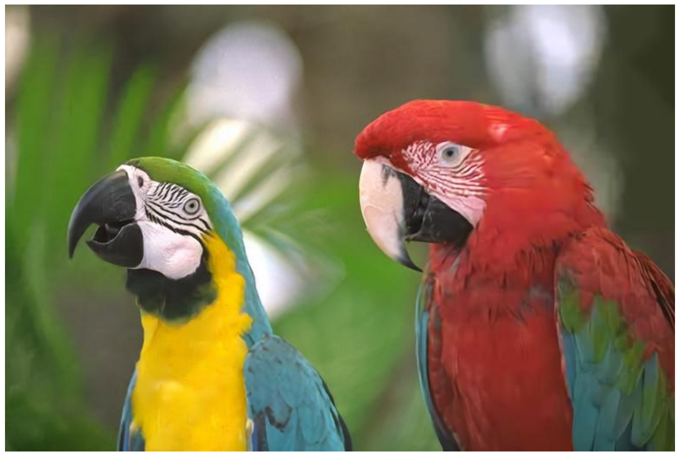
Fig. 9: An example to show the capacity of CDnCNN-3 for three different tasks. The input image is composed of following noisy images - noise level 15 (upper left) and 25 (lower left), bicubically interpolated low resolution images with scaling factor 2 (upper middle) and 3 (lower middle), JPEG images with quality factor 10 (upper right) and 30 (lower right). The white lines in the input image are just to distinguish the different regions, and the residual image has been normalized in the range of [0,1] for visualization. Even with different types of corruptions, the restored image looks natural without any artifact.



(a) Input Image

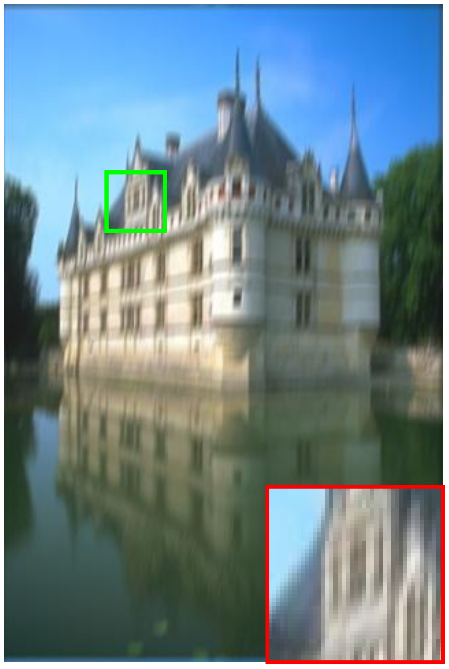


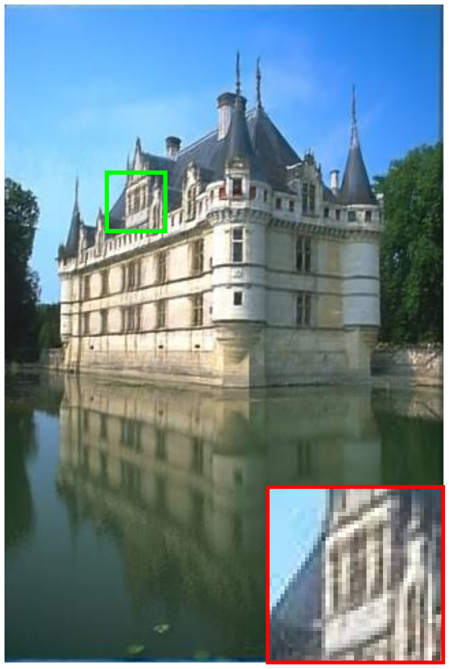
(b) Output Residual Image



(c) CDnCNN-3

Fig. 10: An example to show the capacity of CDnCNN-3 for three different tasks. The input image is composed of following noisy images - noise level 15 (upper left) and 25 (lower left), bicubically interpolated low resolution images with scaling factor 2 (upper middle) and 3 (lower middle), JPEG images with quality factor 10 (upper right) and 30 (lower right). The white lines in the input image are just to distinguish the different regions, and the residual image has been normalized in the range of [0,1] for visualization. Even with different types of corruptions, the restored image looks natural without any artifact.


(a) Blurred Image


(b) CDnCNN-Deblur

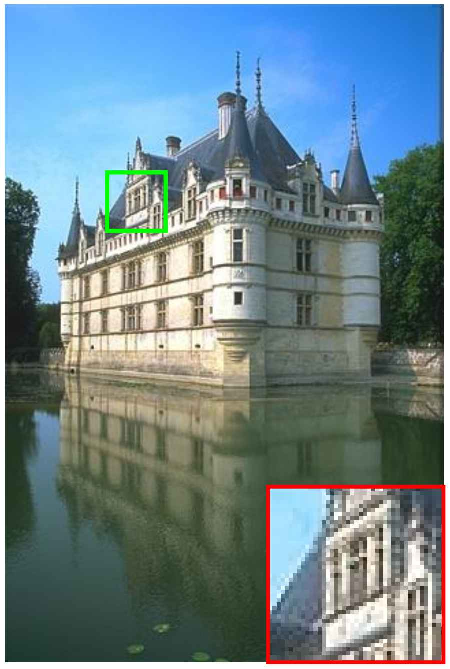
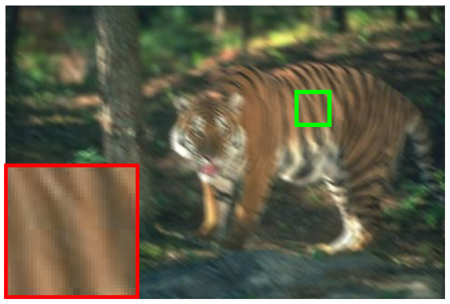
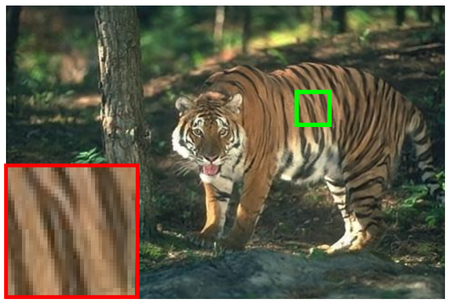

(c) Ground Truth

Fig. 11: Image Deblurring on one image from BSD68 Dataset with blur length 10


(a) Blurred Image


(b) CDnCNN-Deblur

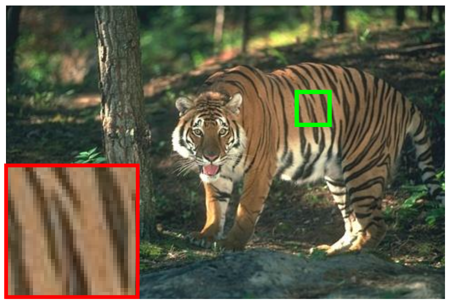
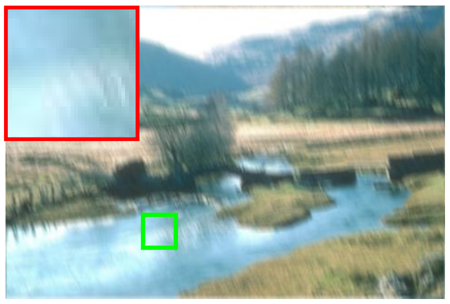
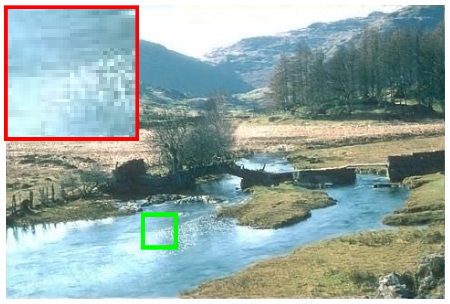

(c) Ground Truth

Fig. 12: Image Deblurring on one image from BSD68 Dataset with blur length 10


(a) Blurred Image


(b) CDnCNN-Deblur

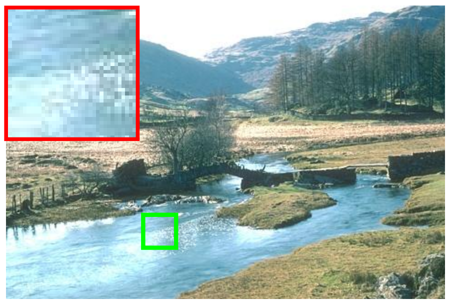

(c) Ground Truth

Fig. 13: Image Deblurring on one image from BSD68 Dataset with blur length 10

References

[1] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng,and Lei Zhang, “Beyond a Gaussian Denoiser: Residual Learningof Deep CNN for Image Denoising,” IEEE Transactions on Image Processing, vol. 26, no. 7, July 2017.

[2] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero P. Simoncelli, ”Image Quality Assessment: From Error Visibility to Structural Similarity,” IEEE Transactions on Image Processing, vo. 13, no. 4, April 2004.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, ”Deep Residual Learning for Image Recognition,” 2016 IEEE Conference on Computer Vision and Pattern Recognition.

[4] S. Ioffe, C. Szegedy, ”Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in Proc. Int. Conf. Mach. Learn., 2015.

[5] Kostadin Dabov *et.al*, ”Image denoising by sparse 3D transform-domain collaborative filtering,”IEEE Transactions on Image Processing, vol. 16, no. 8, August 2007

[6] Michael Elad and Michal Aharon , ”Image Denoising Via Sparse and Redundant Representations Over Learned Dictionaries,”IEEE Transactions on Image Processing, vol. 15, no. 12, December 2006

[7] Berekeley Segmentation DataSet - https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

[8] Kai Zhang, Wangmeng Zuo, and Lei Zhang, ”FFDNet: Toward a Fast and Flexible Solutionfor CNN-Based Image Denoising,” IEEE Transactions on Image Processing, vol. 27, no. 9, September 2018.