Digital Image Denoising

Thomas Liu
Stanford University
tcliu@stanford.edu

Abstract

Image denoising is an image processing technique with applications in computer vision and photography. In this project, we frame digital image denoising as a machine learning problem and employ various techniques toward image denoising. We begin with linear regression and explore its strengths and weaknesses. We briefly consider the suitability of principal component analysis for image denoising. Using deep learning, we implement a more complex model and empirically show that it performs the best of the three models.

1. Introduction

Recent developments in technology have made denoising of digital images more relevant than ever. The trend towards more integrated cameras results in smaller and noisier image sensors. Safe autonomous driving requires imaging systems that perform well under all lighting conditions, even in the middle of the night. High performance denoising systems need to not only produce quality images but also be computationally efficient as further processing may be applied to the image afterwards.

In this project, we take a data-driven approach to the task of digital image denoising and explore the relevance of various machine learning methods. This project focuses primarily on post-processing of still images, and the objective is to maximize image quality with less concern for computation complexity or throughput. At its core, image denoising can be framed as a high dimensional regression problem, so we begin by implementing least squares linear regression. We evaluate the effectiveness of unsupervised learning methods by applying principal component analysis (PCA) to image denoising. Inspired by the success of deep learning techniques in the field of computer vision, we finally consider a convolutional neural network (CNN) based approach to image denoising.

We generate a custom dataset for this project from image data produced by a real image sensor. The primary sources of noise in digital images are generally known and well understood [1], enabling one to create a synthetic dataset from noise-free images. However, knowing the noise models beforehand allows one to craft a system tailored to specifically address those models. Using real training images results in a system that can better generalize to real world performance, though it does introduce sources of error other than noise that would not be present in a synthetic dataset.

2. Related work

Image denoising is a well-studied task and a large body of prior work exists for it. Classical methods for image denoising typically do not involve learning. A common method is to convolve the image with a low-pass filter kernel such as the Gaussian kernel. Though effective, low-pass filtering also results in blurring of the image which can be undesirable. Nonlinear filtering methods such as the median filter [2] are an improvement over weighted averaging in that they are better at preserving edges and image detail.

Current state-of-the-art methods such as BM3D [3] and EPLL [4] are nonlocal self-similarity (NSS) models. The general NSS procedure is to first divide the image into small patches. Similar patches are grouped together and the denoised patch is taken as a weighted average of patches that resemble it. In a way, these methods are like locally weighted linear regression; the primary difference being that local is defined in terms of patch similarity rather than spatial distance.

We do not challenge the effectiveness of such methods; rather, we would like to see if alternate methods can achieve similar denoising performance. The application of convolutional neural networks to computer vision tasks has proven to be very successful in recent years, and this may extend to the task of image denoising as well. Recent works in this domain include the use of CNNs for the task of image deartifacting [5] as well as single image super-resolution [6]. Both tasks are conceptually related image denoising; ultimately, they can be interpreted as image denoising just with different models for noise.

3. Dataset

The dataset consists of 20 scenes captured with a digital camera. Each scene is captured six times under different ISO sensitivity settings with everything else kept the same (aside from shutter speed which the camera automatically adjusts). ISO 100 is chosen as the ground truth image; it is very close to noise-free since the long shutter speed integrates away shot (Poisson) noise due to the photon nature of light and amplification of the raw signal is disabled. Increasing the sensitivity adds more noise to the image, and we choose ISO settings of 400, 1600, 6400, 25600 and 102400 as the noisy images.

Figure 1. A sample scene from the dataset

After capture, the images are imported from raw format with minimal post-processing (just demosaicing to reconstruct the original image). Images from the same scene are realigned to correct for slight movement of the camera. Residual errors at this stage include rotational and translational misalignment, small differences in exposure and small shifts in color temperature caused by the added noise. Although it is possible to correct for these errors, we expect that a robust denoising system should be able to handle these errors as long as it is not systematic.

Each image is relatively large (7952 by 5304 pixels) which may cause computation resource issues if the whole image were to be processed at once. To alleviate this issue, each image is divided into sub-images of 512 by 512 pixels. Doing so generates 150 sub-images per image, or 750 noisy/noise-free pairs per scene. We start with a preliminary test/dev/train split of 73.3%/13.3%/13.3%, partitioning across sub-images within a scene. In total, the dataset contains 11000 training, 2000 validation and 2000 test image pairs.

Figure 2. Images from the sample scene showing progressively more noise. From top to bottom, left to right: ISO 100, 400, 1600, 6400, 25600, 102400.

4. Methods

In this section, we begin by discussing parts of the framework that are common to all models. We then go into more detail about each model implemented for this project.

4.1. Normalization

Before each image is used in training or evaluating a model, the image is normalized so that pixel intensities of take values between 0.0 and 1.0 instead of 0 and 255.

4.2. Loss

Since we frame image denoising as a linear regression problem, a logical choice for loss is squared loss. In particular,

$$l(\theta) = \sum_{y=1}^{H}\sum_{x=1}^{W}\sum_{c\in R,G,B}\left(I^{(c)}(x,y) - \hat{I}_\theta^{(c)}(x,y)\right)^2$$

where I is the ground truth image and $\hat{I}_\theta$ is the denoised image. Some models may add an additional term to the loss such as regularization between different components of θ, but squared loss should serve as the base of every loss function.

4.3. Evaluation metrics

When dealing with noisy signals, the standard evaluation metric is signal-to-noise ratio (SNR). A variant of this metric known as peak signal-to-noise ratio (PSNR) is often used to evaluate visual quality of images. This metric is defined in terms of mean squared error [7]:

$$MSE = \frac{1}{HW}\sum_{y=1}^{H}\sum_{x=1}^{W}\sum_{c\in R,G,B}\left(I^{(c)}(x,y) - \hat{I}_\theta^{(c)}(x,y)\right)^2$$

$$PSNR = -10\log_{10}(MSE)$$

Since PSNR uses a form of squared loss in its computation, it legitimizes the choice of squared loss as the basic loss function. We choose not to optimize directly for PSNR as having a separate loss function gives us more flexibility in engineering it (for example, adding regularization).

The other metric is structural similarity (SSIM), a perceptual metric for visual quality [8]. Rather than looking at absolute differences, this metric compares luminance, contrast and structure between corresponding windows of two images and returns an index between -1 (least similar) and 1 (most similar).

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

where μ_x, μ_y, σ_x^2, σ_y^2 and σ_{xy} are mean, variance and covariance of the windows respectively and c_1 and c_2 are small constants.

4.4. Linear regression model

We begin our experiments by implementing a linear regression model that uses only pixel intensities as features. For each pixel in the image, we extract an 11x11 region around that pixel, reshape it into a 363-dimensional feature vector, then use linear regression to predict the output for

each of the three color channels.

This operation can be more compactly expressed in terms of 2D convolution, if we let the kernel size be 11x11 with three input and output channels and a stride of one. For this model, we implement linear regression as a convolution operation as it is expected to be more computationally efficient than implementing the loop ourselves.

4.5. PCA: best rank-*k* approximation

Next, we would like to see if dimensionality reduction is a viable approach for image denoising. Although dimensionality reduction has more in common with lossy compression than with denoising, we may be able to find images where the first *k* principal components correspond to the noiseless image.

To perform the best rank-*k* approximation, we first treat each color channel of the noisy image as a 512x512 matrix $\boldsymbol{I_c}$. Singular-value decomposition results in the factorization $\boldsymbol{I_c} = \boldsymbol{U\Sigma V}^T$ where $\boldsymbol{\Sigma}$ is a diagonal matrix. From this, we construct the low-dimensional approximation $\hat{\boldsymbol{I}}_{\boldsymbol{c}} = \sum_{i=1}^{k} \sigma_i u_i v_i^T$.

The remaining task is to select a good value for k which, as one may expect, varies from image to image. We define the optimal *k* as the one yielding the highest PSNR and for our initial experiments we assume this value is known for all images. A more realistic implementation could start by characterizing the training set for optimal *k*, then training a regression model (e.g. a simple CNN) to predict the optimal *k* given a noisy image as the input. We choose to stop at this point because even though PCA performs reasonably well on the metrics, the resulting image quality is too poor for it to be used for denoising.

4.6. Denoising CNN

The final model implemented for this project is based on the work on denoising CNNs (DnCNN) by Zhang et al. [9] Despite its relatively straightforward feed-forward convolutional neural network architecture, the model achieves state-of-the-art performance. The authors attribute their results to three factors: residual learning, batch normalization and very deep network architectures.

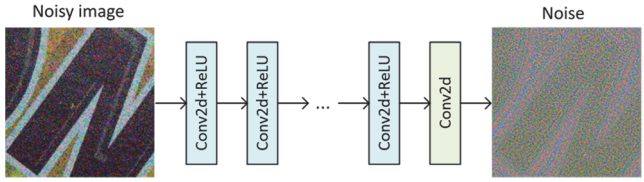

Figure 3. Architecture of the implemented DnCNN model

4.6.1 Residual learning

With residual learning, the CNN learns to predict the noise in the image which is claimed to improve learning and overall accuracy compared to predicting the noise-free image. Implementing residual learning requires changes to both the output of the system and the loss function. We define the noise image E as the ground truth image I subtracted from the noisy image K, and the CNN outputs its prediction of the noise $\hat{E}$. Thus, the modified loss function becomes:

$$l(\theta) = \sum_{y=1}^{H} \sum_{x=1}^{W} \sum_{c \in R,G,B} \left(E^{(c)}(x,y) - \hat{E}_{\theta}^{(c)}(x,y) \right)^2$$

4.6.2 Batch normalization

Batch normalization, a regularization method where the inputs to each layer are first normalized [10], is also touted to improve learning and performance. For our implementation, we choose to not include batch normalization layers. This decision is mainly motivated by hardware limitations on model size, as adding batch normalization effectively doubles the size of the model. A smaller sub-image size (e.g. 128x128) would allow for larger models and we can consider this for future works.

4.6.3 Very deep network architectures

There are three main design parameters that define the DnCNN architecture: the network depth D, the number of hidden units H and the kernel size K. The authors of the original work fix H=64 and K=3 and explore very deep networks starting at 17 layers. Due to model size restrictions, we choose to explore models with fewer layers and instead evaluate the effects of changing H and K.

4.6.4 Training details

We initialize all weights with the Xavier method [11] and all biases to zero. We use Adam optimizer [12] for training with learning rates ranging from 0.0001 to 0.001. Minibatch sizes range from 5 to 25, depending on model complexity. For regularization, we use early stopping and train each model for a maximum of 30 epochs. Each epoch takes anywhere from 20 minutes to 2 hours to train on a single GTX 1080 GPU, depending on model complexity.

5. Results

In this section, we quantitatively and qualitatively evaluate the results produced by each model. Table 1 summarizes PSNR and SSIM on the test set for linear regression and PCA as well as selected configurations of the DnCNN model.

5.1. Linear regression model

The linear regression model performs reasonably well (considering its simplicity), especially on the noisier images. As a baseline, we include results from median filtering which returns the median pixel value from a window of the same size. Though the linear regression model falls short compared to the median filter, we think this is a fair tradeoff given the increased computational complexity of finding the median. Looking at the images, the linear regression model is better at preserving the noise in the image; looking at the learned weights, the center pixel has a relatively high weight.

Model	D	H	K	PSNR						SSIM					
				400	1600	6400	25k6	102k	Avg.	400	1600	6400	25k6	102k	Avg.
Identity				29.1	24.7	19.3	13.4	9.3	19.2	0.933	0.823	0.600	0.319	0.165	0.568
OLS			11	27.1	26.9	25.4	21.6	16.9	23.6	0.945	0.932	0.906	0.824	0.684	0.858
Median			11	29.7	28.8	26.2	21.7	16.9	24.7	0.939	0.935	0.917	0.854	0.730	0.875
PCA				31.4	29.2	25.7	21.0	16.7	24.8	0.962	0.927	0.877	0.794	0.707	0.854
DnCNN	8	64	3	32.1	30.9	28.4	24.6	20.3	27.3	0.966	0.951	0.925	0.867	0.792	0.900
DnCNN	8	64	5	32.3	30.9	29.1	**26.1**	**21.8**	28.0	0.969	0.951	0.932	**0.904**	**0.851**	**0.921**
DnCNN	8	64	7	**32.4**	**31.3**	**29.3**	26.0	**21.8**	**28.2**	**0.970**	**0.955**	**0.937**	0.901	0.844	**0.921**
DnCNN	14	32	3	31.8	30.6	28.3	24.8	20.1	27.1	0.962	0.948	0.924	0.880	0.798	0.902
DnCNN	14	32	5	31.7	30.5	28.3	24.7	20.1	27.1	0.962	0.942	0.919	0.862	0.764	0.890
DnCNN	14	32	7	31.9	30.9	28.8	25.3	21.0	27.6	0.964	0.951	0.929	0.891	0.827	0.912

Table 1: Summary of evaluation metrics on the test set at different ISO sensitivity settings

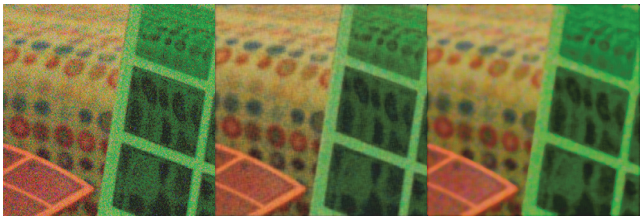

Figure 4. Denoised images with ISO sensitivity of 25600. From left to right: noisy, linear regression, median. Linear regression model generates images with higher residual noise.

An unexpected observation is that PSNR of denoised images is lower than that of the noisy images when the noise content is low (ISO400). Though the linear regression model preserves more detail than the median filter, there is a slight but noticeable color shift in the denoised images that it produces. The fact that linear regression outperforms median filtering on the SSIM metric adds credibility to this hypothesis and highlights the advantages of SSIM in cases like this.

Figure 5. Denoised images with ISO sensitivity of 400. From left to right: noisy, linear regression, median. Linear regression model retains more detail but shifts colors.

5.2. PCA

Based on the metrics alone, one would think that low-rank approximation is a reasonable method for denoising images. On the PSNR metric it performs similarly to the median filter and even outperforms it on images with low noise. (One caveat is that we assume the optimal k is known, so achievable results will be somewhat lower.)

Qualitative analysis of the resulting images reveals the weaknesses of this approach. Low-rank approximation works best when there is a clear separation between image and noise components; this occurs when the image is low in detail and/or out of focus and noise levels are low, as seen in the first row of Figure 6. The middle row shows an example of an image with moderate detail and noise, and noise components rise in rank which degrades image quality. The final row shows the result with high detail and noise; the image is severely degraded by compression artifacts and significant amounts of noise remain. Hence, we cannot recommend this approach for general purpose denoising.

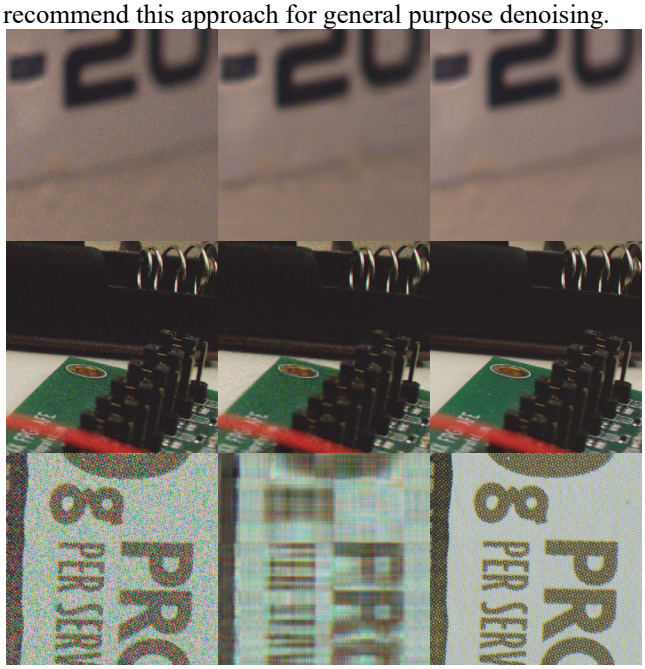

Figure 6. From left to right: noisy, PCA denoised and ground truth. Top image is ISO1600 and k=18, middle image is ISO6400 and k=40, bottom image is ISO102400 and k=5.

What explains the decent qualitative results achieved by this model? A quick analysis of the dataset shows that a large proportion of images fall in the out of focus, low detail category.

5.3. DnCNN

As one may expect, the denoising CNN model easily outperforms the other two models, even with as few as three layers. At eight layers, we see a 3-dB improvement over the linear regression model which translates to a factor of two reduction in noise.

To find the optimal network architecture, we perform coordinate descent on the design parameters starting with a network depth D of two layers. Increasing the number of layers steadily increases the performance of the model. At ten layers, we halve the number of hidden units to enforce an arbitrary minimum minibatch size of 10 and see a decrease in average PSNR of 0.4 dB. Further increasing the network depth, we find that a network with 14 layers and 32 hidden units per layer performs roughly on par with a network with 8 layers and 64 hidden units per layer. We hypothesize that "network power" expressed as the product of layer count and hidden unit count is a better predictor of denoising ability than layer count alone.

Next, we examine the effects of increasing the kernel size K. Increasing K from 3 to 5 and then to 7 is generally beneficial, but kernel sizes of 9 and greater end up being detrimental. One hypothesis is that this is a side effect of zero padding. The input of each convolutional layer is padded with zeros to ensure that the output dimensions match the input dimensions. As kernel size increases, more pixels in the image are affected by padding and the network allocates more resources towards compensating for this effect.

From the results, we conclude that a network with eight layers, 64 hidden units and a kernel size of either 5 or 7 is the best configuration given our model size constraints. Subjective analysis of denoised images shows that denoised image quality is excellent up to ISO25600 and acceptable at ISO102400.

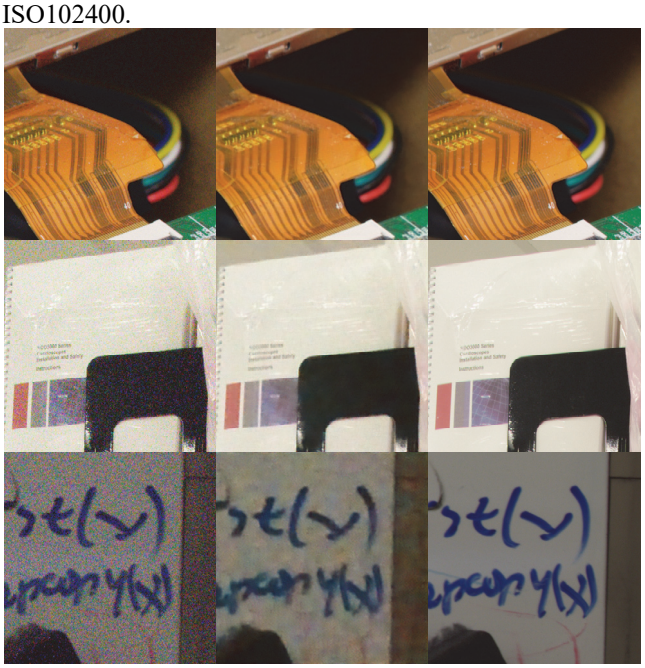

Figure 7. From left to right: noisy, DnCNN denoised and ground truth. Top image is ISO1600, middle image is ISO25600 and bottom image is ISO102400.

Figure 7 shows a sample images produced by the DnCNN model. At low levels of noise, the most noticeable flaws are a decrease in overall sharpness in the denoised image as well as removal of dust or grain that the CNN misinterprets as noise. These effects set an upper limit on the quantitative results achievable by the denoiser. At high levels of noise, blotchy artifacts left behind by incomplete denoising are immediately noticeable in the denoised image. In some patches, a small amount of crosshatch or halftone patterning not present in the original image can be observed. This hints that a small amount of overfitting is occurring; the network is memorizing patterns present in a subset of the data (e. g. images of cereal boxes) and copying them to the output.

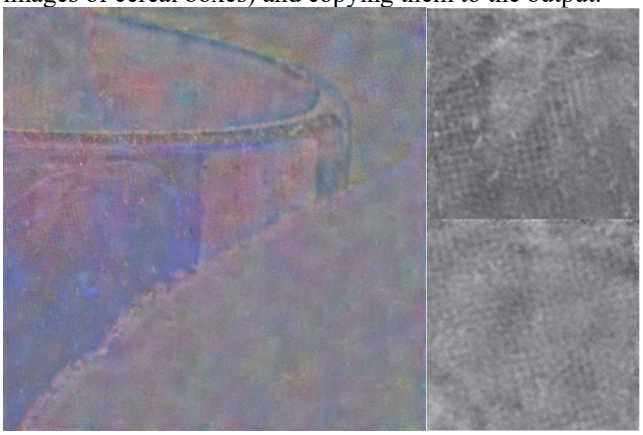

Figure 8. Prediction error computed by subtracting the ground truth image from the denoised image. Detail of crosshatch and halftone pattern on right, contrast and saturation adjusted to improve clarity.

6. Conclusion

In this project, we used machine learning techniques to build and evaluate three models for digital image denoising. The first method uses linear regression to learn optimal weights for a low-pass filtering approach to image denoising. Results show that this method performs reasonably well as a starting point for our experiments. The second method employs low-rank approximation using PCA and we conclude that this method is not suitable for denoising when image complexity and/or noise levels are high. The final method implements a denoising CNN which soundly outperforms the other two approaches, achieving over 10 dB increase in PSNR on noisier images.

There are many options available to further expand upon this project. A quick way to improve performance of the linear regression and DnCNN models could be to train separate weights for each ISO sensitivity setting. This information is often included in image metadata; lacking it, one could build a regression model to predict noise level from the image. Another would be to apply denoising before demosaicing since denoising performance often improves when chromatic noise and luminance noise are handled separately. Finally, we would like to augment the dataset with images captured with a wide variety of image sensor sizes and technologies since noise characteristics could vary significantly.

7. References

[1] Image Noise, 2017. en.wikipedia.org/wiki/Image_noise

[2] T. Huang, G. Yang and G. Tang. “A fast two-dimensional median filtering algorithm”, *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 27, no. 1, pp. 13–18, 1979.

[3] K. Dabov, A. Foi, V. Katkovnik and K. Egiazarian. “Image denoising with block-matching and 3D filtering,” *Proc. SPIE Electronic Imaging '06*, no. 6064A-30, San Jose, California, USA, January 2006.

[4] D. Zoran and Y. Weiss. “From learning models of natural image patches to whole image restoration”, *2011 International Conference on Computer Vision*, pp. 479–486, November 2011.

[5] P. Svoboda, M. Hradis, D. Barina and P. Zemcik. “Compression Artifacts Removal Using Convolutional Neural Networks”, arXiv:1605.00366 [cs.CV], 2016.

[6] C. Dong, C.C. Loy, K. He and X. Tang. “Image Super-Resolution Using Deep Convolutional Networks”, arXiv:1501.00092 [cs.CV], 2016.

[7] Peak signal-to-noise ratio, 2017. en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

[8] Z. Wang, A.C. Bovik, H.R. Sheikh and E.P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". *IEEE Trans. Image Process*. vol. 13, no. 4, pp. 600–612, 2004.

[9] Bilateral filter, 2017. en.wikipedia.org/wiki/Bilateral_filter

[10] S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, arXiv:1502.03167 [cs.LG], 2015.

[11] X. Glorot and Y. Bengio. “Understanding the difficulty of training deep feedforward neural networks”, AISTATS, 2010.

[12] D.P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”, ICLR, 2015.