Supplementary Material for:

PieAPP: Perceptual Image-Error Assessment through Pairwise Preference

Ekta Prashnani* Hong Cai* Yasamin Mostofi Pradeep Sen
University of California, Santa Barbara
{ekta,hcai,ymostofi,psen}@ece.ucsb.edu

Contents

*Joint first authors.

This project was supported in part by NSF grants IIS-1321168 and IIS-1619376, as well as a Fall 2017 AI Grant (awarded to Ekta Prashnani).

1 Performance Comparisons with Additional IQA Methods and Datasets (Paper Sec. 5)

In the main paper, we compare the performance of PieAPP on our proposed test set (which is disjoint from the training set in terms of both the reference images and the distortion types) against popular or state-of-the-art IQA methods. In this section, we compare PieAPP against 11 additional popular or state-of-the-art IQA methods on our proposed test set (similar to Sec. 5.2 of the main paper). Furthermore, we test the efficacy of our deep convolutional neural network (DCNN) architecture by training and testing the error-estimation function (i.e., $f(A, R; \theta)$) of our pairwise-learning framework on 2 additional existing IQA datasets, TID2008 [1] and LIVE [2] (in the same manner as the training and testing on TID2013 and CSIQ; Sec. 5.3 of main paper).

Brief descriptions of the IQA methods considered in the comparisons (both in the main paper and this supplementary) are also provided. In the descriptions, we only highlight the key characteristics of the IQA methods. More in-depth details can be found in the corresponding references.

METHOD	KRCC		PLCC	SRCC
	$\tilde{p}_{AB} \in [0, 1]$	$\tilde{p}_{AB} \notin [0.35, 0.65]$		
MAPE	0.233	0.278	0.170	0.290
MRSE	0.185	0.211	0.135	0.219
NQM	0.216	0.234	0.508	0.246
IFC	0.165	0.178	0.230	0.198
VIF	0.179	0.192	0.250	0.212
VSNR	0.242	0.274	0.286	0.281
MAD	0.270	0.301	0.231	0.304
RFSIM	0.217	0.246	0.368	0.247
GSM	0.324	0.376	0.356	0.378
SR-SIM	0.296	0.356	0.352	0.358
MDSI	0.280	0.336	0.349	0.350
MAE	0.252	0.289	0.302	0.302
RMSE	0.289	0.339	0.324	0.351
SSIM	0.272	0.323	0.245	0.316
MS-SSIM	0.275	0.325	0.051	0.321
GMSD	0.250	0.291	0.242	0.297
VSI	0.337	0.395	0.344	0.393
PSNR-HMA	0.245	0.274	0.310	0.281
FSIMc	0.322	0.377	0.481	0.378
SFF	0.258	0.295	0.025	0.305
SCQI	0.303	0.364	0.267	0.360
DOG-SSIMc	0.263	0.320	0.417	0.464
Lukin et al.	0.290	0.396	0.496	0.386
Kim et al.	0.211	0.240	0.172	0.252
Bosse et al. (NR)	0.269	0.353	0.439	0.352
Bosse et al. (FR)	0.414	0.503	0.568	0.537
Our method (PieAPP)	**0.668**	**0.815**	**0.842**	**0.831**

Table 1: Additional comparisons for assessing the performance of our approach compared to existing IQA methods on our test set (which is disjoint from the training set in terms of both the reference images and the distortion types). PieAPP significantly improves upon the state-of-the-art methods, which do not perform well because this test set contains many different (and complex) distortions not commonly found in standard IQA datasets.

1.1 Comparisons with additional IQA methods on the proposed test set

Table 1 shows the performance comparisons on our proposed test set (consisting of 4200 pairwise comparisons obtained from exhaustively labeling all possible pairwise comparisons for 15 distorted copies for each test reference image; see Sec. 4.2 and 5.2 of the main paper)[1]. The methods which have already been listed in the paper are shown in the lower section of the table, while the additional methods are shown in the upper section. The performance of PieAPP is shown at the very end of the table (highlighted in bold). We emphasize that none of these test images and test distortions have been seen during training of PieAPP,

[1]For an existing IQA method, its PLCC on our test set is computed after fitting its predicted scores to the ground-truth scores via a nonlinear regression [2].

and therefore, enable us to understand the generalizability of our proposed method and existing methods. We follow the same evaluation strategy as discussed in Sec. 5.2 of the main paper. As can be seen, PieAPP outperforms all existing IQA methods.

1.2 Additional comparisons on the existing IQA datasets

As stated in the main paper, we retrain and evaluate the error-estimation function of our pairwise-learning framework (i.e., $f(A, R; \theta)$) on existing IQA datasets using the MOS labels of each dataset as the ground truth. This enables a direct comparison against existing methods using the numbers reported by these methods in their papers after training and testing on these datasets. We train our error-estimation function directly on the MOS labels as existing datasets do not provide probabilistic pairwise labels. However, this does not change our learning architecture of f or its number of parameters. In the main paper, we report the performance of our architecture on the two largest (and least overlapping) IQA datasets, CSIQ [3] and TID2013 [4]. Here, we report the performance on two additional popular IQA datasets, LIVE [2] and TID2008 [1] (see Table. 2). As can be seen, our proposed architecture for $f(A, R; \theta)$ outperforms or gives comparable performance to existing state-of-the-art IQA methods.

METHOD	LIVE [2]			TID2008 [1]			CSIQ [3]			TID2013 [4]		
	KRCC	PLCC	SRCC	KRCC	PLCC	SRCC	KRCC	PLCC	SRCC	KRCC	PLCC	SRCC
MAPE	0.841	0.892	0.934	0.361	0.430	0.448	0.587	0.607	0.706	0.477	0.579	0.586
MRSE	0.842	0.621	0.935	0.394	0.477	0.487	0.619	0.568	0.754	0.501	0.631	0.611
NQM	0.741	0.912	0.909	0.461	0.614	0.624	0.564	0.743	0.740	0.474	0.690	0.643
IFC	0.758	0.927	0.926	0.424	0.734	0.568	0.590	0.838	0.767	0.394	0.554	0.539
VIF	0.828	0.960	0.964	0.586	0.808	0.749	0.754	0.928	0.920	0.515	0.772	0.677
VSNR	0.762	0.923	0.927	0.534	0.682	0.705	0.625	0.800	0.811	0.508	0.740	0.681
MAD	0.842	0.968	0.967	0.644	0.831	0.834	0.797	0.950	0.947	0.604	0.827	0.781
RFSIM	0.782	0.935	0.940	0.678	0.864	0.868	0.764	0.918	0.930	0.595	0.833	0.774
GSM	0.815	0.951	0.956	0.660	0.842	0.850	0.737	0.896	0.911	0.626	0.846	0.795
SR-SIM	0.830	0.955	0.962	0.715	0.887	0.891	0.772	0.925	0.932	0.666	0.877	0.851
MDSI	0.840	0.966	0.967	0.752	0.916	0.921	0.813	0.953	0.957	0.712	0.908	0.890
MAE	0.814	0.567	0.936	0.228	0.120	0.321	0.639	0.644	0.813	0.351	0.294	0.484
RMSE	0.812	0.917	0.931	0.187	0.168	0.265	0.617	0.752	0.783	0.327	0.358	0.453
SSIM	0.796	0.945	0.948	0.577	0.773	0.775	0.691	0.861	0.876	0.464	0.691	0.637
MS-SSIM	0.804	0.949	0.951	0.657	0.845	0.854	0.739	0.899	0.913	0.608	0.833	0.786
GMSD	0.856	0.960	0.960	0.727	0.879	0.891	0.812	0.954	0.957	0.634	0.859	0.804
VSI	0.806	0.948	0.952	0.712	0.876	0.898	0.786	0.928	0.942	0.718	0.900	0.897
PSNR-HMA	0.726	0.874	0.872	0.673	0.819	0.847	0.780	0.888	0.922	0.632	0.802	0.813
FSIMc	0.836	0.961	0.964	0.699	0.876	0.884	0.769	0.919	0.931	0.667	0.877	0.851
SFF	0.836	0.963	0.965	0.688	0.882	0.877	0.828	0.964	0.963	0.658	0.871	0.851
SCQI	0.784	0.934	0.941	0.729	0.890	0.905	0.787	0.927	0.943	0.733	0.907	0.905
DOG-SSIMc	0.844	0.966	0.963	0.786	0.939	0.935	0.813	0.943	0.954	0.768	0.934	0.926
Lukin et al.	–	–	–	–	–	–	–	–	–	0.770	–	0.930
Kim et al.	–	0.982	**0.981**	–	0.951	0.947	–	0.965	0.961	–	**0.947**	0.939
Bosse et al. (NR)	–	0.963	0.954	–	–	–	–	–	–	–	0.787	0.761
Bosse et al. (FR)	–	0.980	0.970	–	–	–	–	–	–	0.780	0.946	0.940
Error-estimation f	**0.894**	**0.986**	0.977	**0.822**	**0.956**	**0.951**	**0.881**	**0.975**	**0.973**	**0.804**	0.946	**0.945**

Table 2: Comparison on additional standard IQA databases. Here we show comparisons against LIVE [2], TID2008 [1], CSIQ [3] and TID2013 [4]. For all learning methods, we use the numbers directly provided by the authors (dashes "–" are shown where numbers are not provided). For a fair comparison, we train the error-estimation function of our pairwise-learning framework (i.e., $f(A, R; \theta)$) directly on the MOS labels of each dataset as existing datasets do not provide probabilistic pairwise labels.

1.3 Brief descriptions of the IQA methods included in the paper

In the paper, we have included several state-of-the-art IQA methods, which are **1)** very commonly-used IQA methods (1-4), **2)** recently-proposed IQA methods which have good performance over existing IQA datasets (5-10), and **3)** recent top-performing learning-based IQA methods (11-13). We provide brief descriptions of these methods below.

1. **Mean Absolute Error (MAE):** This method calculates the mean absolute error of the pixel values between the distorted image and the reference image. Note that this method is equivalent to calculating

the L1 error between the reference image and the distorted image, in terms of predicting human binary preference.

2. **Root Mean Squared Error (RMSE):** This method calculates the square root of the mean squared error of the pixel values between the distorted image and the reference image. Note that this method is equivalent to calculating the L2 error or the Peak Signal-to-Noise Ratio (PSNR) between the reference image and the distorted image, in terms of predicting human binary preference.

3. **Structural Similarity Index (SSIM) [5]:** This method utilizes the structural image similarity between the reference image and the distorted image.

4. **Multi-Scale SSIM (MS-SSIM) [6]:** This is a multi-scale extension of SSIM.

5. **PSNR-HMA [7]:** This is an improved version of PSNR that takes into account the contrast sensitivity function, between-coefficient contrast masking of DCT basis functions, mean shift, and contrast changing.

6. **Feature Similarity Index (FSIMc) [8]:** This method utilizes the phase congruency and the image gradient magnitude as features to compute the similarity between the reference image and the distorted image.

7. **Spare Feature Fidelity (SFF) [9]:** This method transforms images into sparse representations based on a trained sparse feature detector. It then computes the feature similarity and luminance correlation between the reference image and the distorted image. This algorithm is motivated by the fact that sparse coding can provide a good description of an important part of the Human Visual System (HVS).

8. **Gradient Magnitude Similarity Deviation (GMSD) [10]:** This method utilizes the pixel-wise gradient magnitude similarity between the reference image and the distorted image to predict the perceptual image quality.

9. **Visual Saliency Induced Index (VSI) [11]:** This method uses the visual saliency to compute the local quality map of the distorted image. It then uses the visual saliency again to weigh the importance of the local regions, when pooling the local quality scores.

10. **Structural Contrast Quality Index (SCQI) [12]:** This method primarily uses the structural contrast index, which can well characterize local and global visual quality perceptions for various image characteristics with structural-distortion types. In addition, features that capture contrast sensitivity and chrominance component variation are also used.

11. **Pei et al. [13]:** This learning-based method first extracts features from several difference of Gaussian frequency bands of the image, which mimics the HVS. The features are then nonlinearly combined using the Random Forest regression model to predict the MOS of an image. This algorithm is representative of the class of learning-based methods that compute image features directly from the image pixels and utilize a regression model. This methodology is different from the metric-fusion framework (e.g., Lukin et al.) which utilizes the quality scores provided by existing IQMs, as opposed to directly processing the image.

    As the trained model is not publicly available, we have obtained the feature extraction codes from the authors and re-produced the model based on the details in the paper. Our trained model has matched the reported performance in the cross-validation experiments on TID 2013, where our RMSE is 0.4383 (reported: 0.4433) and our KROCC is 0.7772 (reported: 0.7678).[2] This indicates that our training is implemented correctly. We thus train the model on the entire TID2013 dataset and use this trained model for performance comparison.

[2]This performance is for DOG-SSIMc, which is the best model reported by Pei et al. [13]

12. **Lukin et al. [14]:** This is a neural network-based approach that nonlinearly combines the scores computed by several existing IQMs to predict the Mean Opinion Score (MOS) of an image. This algorithm is representative of the class of learning-based methods that perform *metric fusion*. More specifically, these algorithms use the scores of existing IQMs as the input to a learning system, which is then trained to predict the MOS of an image.

    The model used for comparison is trained on TID 2013, as is done in [14].[3]

13. **Bosse et al. [15]:** This is a Deep Convolutional Neural Network (DCNN)-based approach. It utilizes a DCNN to extract features from a number of image patches. The features are then nonlinearly combined (using fully-connected layers) to predict the MOS of an image. Two versions of the proposed architecture are trained: one for full-reference IQA and another for no-reference IQA. For completeness, we compare against the models released by the authors for both these versions. The models (both FR and NR) used for comparison on our test set are weighted-patch-combination models trained on the TID2013 dataset. For the comparison on existing datasets, we report the numbers published by the authors.[4].

14. **Kim et al. [16]:** This is another Deep Convolutional Neural Network (DCNN)-based approach. It utilizes a DCNN to learn a sensitivity map using the distorted image, and an error map between the reference and the distorted image as input (a version without error map as input feature is also trained, but we choose to compare against the one that utilizes the error map due to better performance). Since this is a recent method, the trained models have not yet been released and so we follow the code released by the authors to perform a 20-fold training on TID2013 (the largest existing IQA dataset with largest variety of distortions) as prescribed and report performance on our test set averaged over these 20 folds. For the comparison on existing datasets, we report the numbers published by the authors.

1.4 Brief descriptions of the additional IQA methods included in the supplementary file

In this section, we briefly summarize the additional IQA methods used for performance comparison in this supplementary file, which include 11 popular model-based methods. These methods are representative of a wide variety of existing IQA methods.

1. **Mean Absolute Percentage Error (MAPE):** This error metric is computed using the following formula:

$$\text{MAPE} = \sum_{i=1}^{N_I} \frac{|I_D(x_i, y_i) - I_R(x_i, y_i)|}{I_R(x_i, y_i) + \epsilon}, \tag{1}$$

    where $I_D(x, y)$ denotes the distorted image, $I_R(x, y)$ denotes the reference image, $\epsilon = 0.01$ is used to avoid division-by-zero, and N_I is the total number of pixels. The numeric range of pixel values is from 0 to 255.

    This is a commonly-used metric in signal processing and it accounts for the HVS's sensitivity to color variations in darker image regions.

2. **Mean Relative Squared Error (MRSE):** This error metric is computed using the following formula:

$$\text{MRSE} = \sum_{i=1}^{N_I} \frac{(I_D(x_i, y_i) - I_R(x_i, y_i))^2}{I_R(x_i, y_i)^2 + \epsilon}, \tag{2}$$

    where $I_D(x, y)$ denotes the distorted image, $I_R(x, y)$ denotes the reference image, $\epsilon = 0.01$ is used to avoid division-by-zero, and N_I is the total number of pixels. The numeric range of pixel values is from 0 to 255.

    This error metric is more sensitive to color variations in darker image regions as compared to RMSE in order to model the HVS.

[3]The trained model is available from the authors via `http://ponomarenko.info/nnmetric.rar`.
[4]The trained model is available from the authors via `https://github.com/dmaniry/deepIQA`

3. **Noise Quality Measure (NQM) [17]:** This methods takes into account the following factors when measuring the visual quality of an image: **1)** variation in contrast sensitivity with distance, image dimensions, and spatial frequency; **2)** variation in the local luminance mean; **3)** contrast interaction between spatial frequencies; and **4)** contrast masking effects.

4. **Information Fidelity Criterion (IFC) [18]:** This method uses a novel information fidelity criterion that is based on natural scene statistics, for assessing the image quality with respect to a reference image.

5. **Visual Information Fidelity (VIF) [19]:** This method proposes an information measure of image content. It first quantifies the information content present in the reference image and then it uses the same information measure to quantify the distorted image. These two information measurements are then used to form the VIF measure, which relates visual quality to relative image information.

6. **Visual Signal-to-Noise Ratio (VSNR) [20]:** This method first computes the contrast thresholds for detecting distortions via wavelet-based models of visual masking and visual summation. When the distortion is visible (above threshold), VSNR utilizes the low-level visual property of perceived contrast and the mid-level visual property of global precedence to measure visual fidelity.

7. **Most Apparent Distortion (MAD) [3]:** This method aims to model two strategies that the HVS uses to determine image quality, which are: **1)** for images containing near-threshold distortions, the image is most apparent, and thus the HVS attempts to look past the image and look for the distortions; **2)** for images containing clearly visible distortions, the distortions are most apparent, and thus the HVS attempts to look past the distortion and look for the image's subject matter.

8. **Riesz-Transform Based Feature Similarity Metric (RFSIM) [21]:** This method utilizes the 1^{st}-order and 2^{nd}-order Riesz transform coefficients of the image as features, which is based on the fact that the HVS perceives an image mainly according to its low-level features.

9. **Gradient Similarity Based Metric (GSM) [22]:** This method puts an emphasis on gradient similarity, which can effectively capture structural and contrast changes. Luminance change is also taken into account in this metric.

10. **Spectral Residual Based Similarity (SR-SIM) [23]:** This method utilizes the spectral residual visual saliency to measure the image quality.

11. **Mean Deviation Similarity Index (MDSI) [24]:** This method utilizes gradient similarity and chromaticity similarity as image features. It then uses a deviation pooling scheme to compute a final quality score based on the proposed image features.

2 Training Details (Paper Sec. 3 and 5.4)

Our training set of 160 reference images and their distorted image pairs (paper Sec. 4.1) is randomly divided into 120 training and 40 validation reference images. For the validation set, the probability labels obtained from human responses are directly used as ground-truth labels without employing ML estimation to obtain some of the missing probability labels (which was done for the training set). Therefore, a total of 67,620 pairs (after ML estimation) are used for training and 7260 pairs (without ML estimation) are used for validation. We use a batch size of 4 during training (i.e., the mean squared error between the predicted and the ground-truth probabilities of 4 image pairs A, B is used for one backward pass through the network). 36 random patches of size 64×64 are sampled from corresponding locations in A, B, and R, and fed through the network to compute s_A and s_B (which are then used to compute the predicted probabilities). For the validation set, 36 patches are sampled on a regular 6×6 grid of overlapping patches. At test time, we ideally want to use all possible patches for computing the overall quality score for an image. However, sampling all possible patches would significantly increase the computation time of our metric even for a single image. We therefore randomly sample 1024 patches during testing of our trained model (which means that with a probability ~ 1, every pixel will be present in at least one sampled patch). We use Adam optimizer [25] for gradient backpropagation with a learning rate of 10^{-4} to minimize the mean squared error between the predicted and the ground-truth probabilities. There is clear evidence of convergence (i.e., negligible change in validation set performance), within 300,000 iterations and hence the training is stopped after that.

2.1 Training existing IQA methods with our pairwise-learning framework

When training the existing DCNN-based IQA methods [15, 16] using our pairwise-learning framework (Sec. 5.4 in the main paper), we use the hyperparameters prescribed by the authors, such as learning rate, batch size, patch sampling strategies, and optimizer choice (all methods used Adam optimizer [25]). Gradient descent is performed on the mean squared error between the predicted and the ground-truth probabilities. The same training-validation split is used for all the trainings (including the training of PieAPP). All the pairwise-preference trainings with existing models converge within 500,000 iterations of training which take at most 2 days on an NVIDIA Titan X GPU using unoptimized TensorFlow code. For each of the trainings, the best performing models are selected based on validation set performance and used for further assessment on our proposed test set (which is disjoint from our proposed training set both in terms of distortion types and reference images), and the two largest existing IQA datasets, TID2013 [4] and CSIQ [3], as reported in the main paper.

3 Patch Sampling Strategies for Training PieAPP (Paper Sec. 3)

We present two analyses to motivate our patch sampling strategy during training.

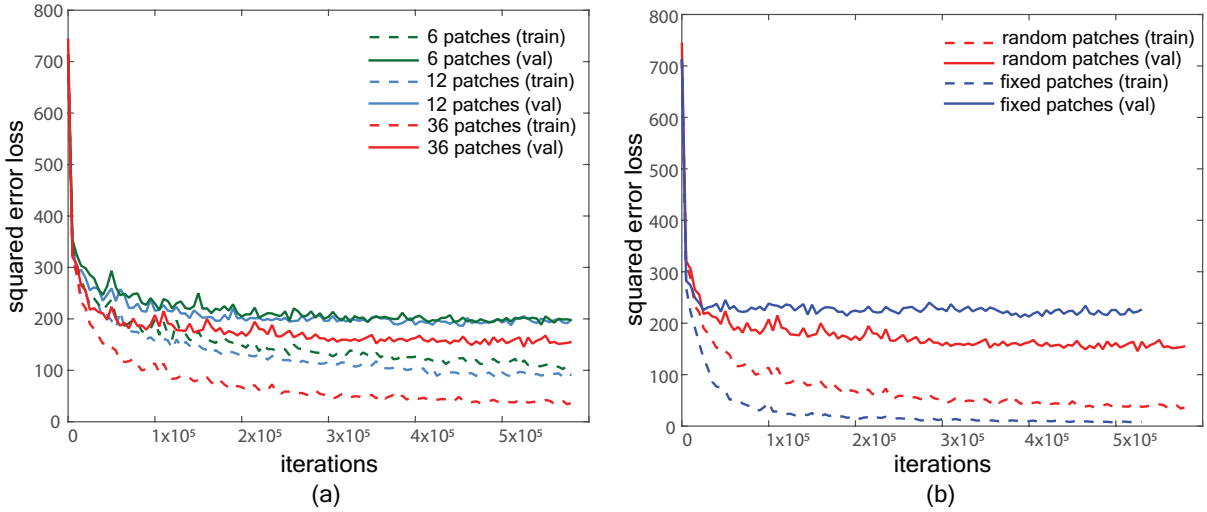


Figure 1: Experiments for patch sampling strategies during training. (a) shows a comparison of the training plots for PieAPP (mean squared error vs. iterations) when the training is performed using sparse random patch samples (6 and 12 patches per image per iteration) and our choice of 36 random patch samples. A better convergence is observed for the training with denser (36) random patch sampling strategy. (b) shows the comparison of the training plots when the training is performed using 36 fixed patch samples and using 36 random patch samples. As can be seen, randomly sampling patches during training has significantly better convergence while the training with fixed samples overfits to training data early on.

Effects of sparse random patch sampling. During training, 36 random patches are sampled during an iteration to compute an image score. We find that if each pixel is a part of at least one patch with a high probability (empirically, we find that a probability ≥ 0.90 to be sufficiently high), the convergence of the training process happens faster compared to when sparser patches are sampled. To demonstrate this, we perform two additional training experiments: 1) with only 6 patches sampled during a training iteration (probability of an image pixel belonging to at least one patch $= 0.32$), and 2) with 12 patches sampled during a training iteration (probability of an image pixel belonging to at least one patch $= 0.54$). We observe that the convergence of the network becomes slower with a reduced number of patch samples (see Fig. 1a). A larger number of patch samples (e.g., 36) can improve convergence but also significantly increase training time and memory requirements. We find 36 patches to be the best balance of training speed and accuracy.

Advantages of random patch sampling. The process of random sampling of patches during training leads to an increase in the variety of samples observed for a single image during an iteration. This in turn enables the network to learn from the given (limited) amount of data for a larger number of iterations thereby leading to improved performance on validation set [15]. To observe this behavior we compare the performance (in terms of mean squared error) of two training sessions: 1) training with 36 patches randomly sampled during each training iteration for every input, and 2) training with 36 patches sampled from fixed positions on a regular 6×6 grid so that every time an image is fed to the network, the same patches are seen during training. We observe that the fixed patch sampling strategy results in significant overfitting to the training set with a validation set performance that is worse than the random sampling case (see Fig. 1b). Therefore, a random patch sampling strategy for training is adopted.

4 Details of Amazon Mechanical Turk Data Collection (Paper Sec. 4.3)

We use Amazon Mechanical Turk (MTurk) to collect human responses for both the training pairs and the unseen test pairs. In each pairwise image comparison, the MTurk user is presented with two distorted images (A and B), along with the reference image (R). The user is asked to select the image that he/she considers to be visually more similar to the reference image. An example interface is shown in Sec. 4.1 of this supplementary file, along with MTurk experiment details.

As mentioned in the paper, it can be prohibitively expensive to accurately estimate the probabilistic human preference p_{AB} for $77,280$ training pairs (paper Sec. 4.3.2). In the paper, we have discussed how to get around this problem by selecting an appropriate number of responses for each comparison and utilizing maximum likelihood (ML) estimation to reduce the number of required queries. We have also established the validity of such an approach in Sec. 5.1 of the paper. Here, we discuss some more details about our MTurk interface, numbers of human responses per pairwise comparison and ML estimation in Sec. 4.2 and 4.3, respectively.

4.1 Details of the MTurk interface

Fig. 2 shows an example MTurk user interface. The user is instructed to select (by clicking the corresponding button below a distorted image) the distorted image from the bottom row that he/she considers to be more similar to the reference image on the top row.

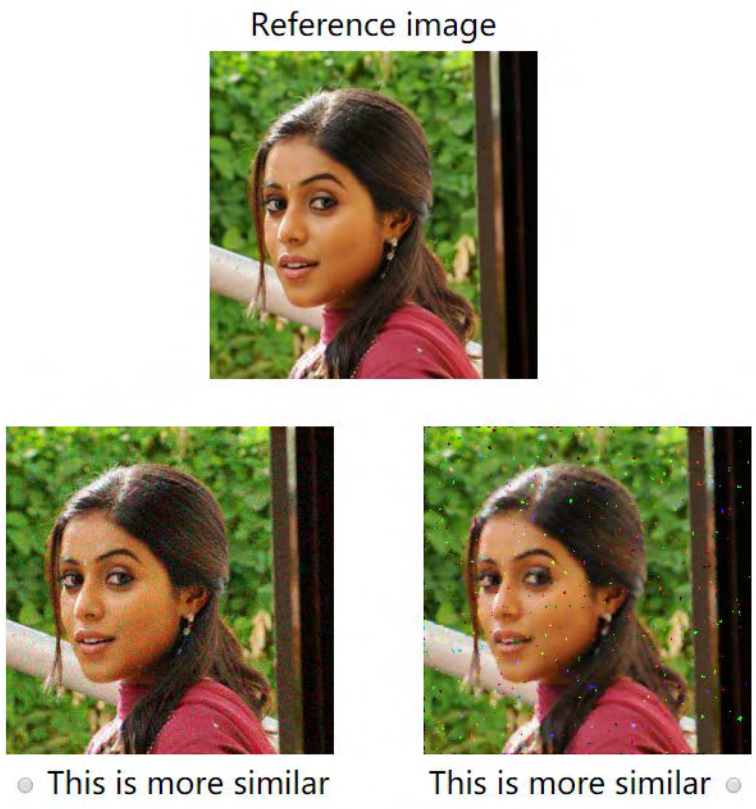


Figure 2: An example MTurk interface - the user selects the distorted image from the bottom row that he/she considers to be more similar to the reference image on the top row.

To ensure reliable data collection, a number of measures have been taken to prevent any potential selection bias: **1)** the left and right buttons are designed to be equally-distanced from the middle; **2)** the positions of the left image and right image are randomized in different queries; **3)** the sequential order in

which the pairwise comparisons appear is also randomized. In addition, we have designed 4 quality control questions in the form of pairwise image comparisons. The quality control questions have obvious answers and are randomly embedded in each MTurk survey. Any submission that fails more than one quality control question will be rejected. Each survey is designed to take approximately 20 min to finish to avoid fatigue. In the actual MTurk experiments, each user is allowed 30 min to finish a survey.

4.2 Details on the number of needed responses per comparison (Paper Sec. 4.3.1)

We model the human response as a Bernoulli random variable with a success probability p_{AB}, which is the probability of a person preferring I_A over I_B. Given n human responses $\nu_i, i = 1, ..., n$, we then have $\tilde{p}_{AB,n} = \frac{1}{n}\sum_{i=1}^{n} \nu_i$, where $\tilde{p}_{AB,n}$ is the p_{AB} estimated using n responses. The Cumulative Distribution Function (CDF) of $|\tilde{p}_{\mathrm{AB,n}} - p_{\mathrm{AB}}|$ is then given as follows:

$$\mathrm{prob}(|\tilde{p}_{\mathrm{AB,n}} - p_{\mathrm{AB}}| \leq \eta) = F((p_{\mathrm{AB}} + \eta)n, n, p_{\mathrm{AB}}) - F((p_{\mathrm{AB}} - \eta)n, n, p_{\mathrm{AB}}), \tag{3}$$

where $F(., n, p_{\mathrm{AB}})$ is the CDF of a Binomial distribution with n trials and a success probability p_{AB}.

To ensure an accurate estimation of p_{AB}, we must choose n such that $\mathrm{prob}(|\tilde{p}_{AB,n} - p_{AB}| \leq \eta) \geq P_{\mathrm{target}}$, for a target P_{target} and tolerance η. Based on the CDF of $|\tilde{p}_{AB,n} - p_{AB}|$ given by Eq. 3, it can be easily confirmed numerically that by choosing $n = 40$ and $\eta = 0.15$, we can achieve $P_{\mathrm{target}} \geq 0.94$. Therefore, we collect 40 responses for each pairwise comparison in the training and test sets.

To further empirically analyze the estimation accuracy of p_{AB}, we collect 100 responses for 630 randomly-selected pairs. We assume that $\tilde{p}_{AB,100} = p_{AB}$. Under this assumption, $E[|\tilde{p}_{AB,40} - p_{AB}|]$ is estimated to be 0.056, which indicates the good accuracy obtained by choosing $n = 40$ for estimating the probabilistic preference. As for estimating the binary preference, we look at the cases where $\tilde{p}_{AB,40} \notin [0.35, 0.65]$, which indicates a strong estimated binary preference.[5] We then have $\mathrm{prob}(p_{AB} > 0.5 \mid \tilde{p}_{AB,40} > 0.65)$ estimated to be 1, and $\mathrm{prob}(p_{AB} < 0.5 \mid \tilde{p}_{AB,40} < 0.35)$ estimated to be 0.992, which confirms that by collecting 40 responses per comparison, $\tilde{p}_{AB,40}$ estimates the binary preference with a high accuracy.

4.3 Details on the statistical estimation of human preference (Paper Sec. 4.3.2)

Collecting 40 MTurk responses for each of the 77,280 pairs of images in the training set is prohibitively expensive. Thus, we use statistical modeling [26] to estimate all the needed human labels based on a subset of the exhaustive pairwise comparison MTurk data. To see how this works, suppose we need to estimate p_A for all the possible pairs of N images (e.g., $N = 15$ in each inter-type group). Assume that the intrinsic quality scores of the images are $s = [s_1, ..., s_N]$. We use the Bradley-Terry model [27] for human responses, which models the probability of choosing one image, I_i, over the other, I_j, as a sigmoid function of the score difference, i.e., $\mathrm{prob}(\mathrm{I}_i > \mathrm{I}_j) = S(s_i - s_j)$, where $S(x) = \frac{1}{1+\exp(-x/\sigma)}$ and $\sigma > 0$.[6] We denote the human responses by a count matrix $C = \{c_{i,j}\}$, where $c_{i,j}$ is the number of times I_i is preferred over I_j. The scores can then be obtained by solving an ML estimation problem [26]:

$$\begin{aligned} s^\star &= \underset{s}{\mathrm{argmax}} \log \prod_{i,j} \mathrm{prob}(\mathrm{I}_i > \mathrm{I}_j)^{c_{i,j}} \\ &= \underset{s}{\mathrm{argmax}} \sum_{i,j} c_{i,j} \log S(s_i - s_j), \end{aligned} \tag{4}$$

which can be solved via gradient descent.

[5] We look at this range since when $p_{AB} \in [0.35, 0.65]$, there does not exist a strong binary preference to estimate and as our estimation of p_{AB} from MTurk data is accurate within $\eta = 0.15$, it is not very meaningful to learn a binary preference within this domain.

[6] The symbol ">" is used to indicate a binary preference in this context, i.e., $X > Y$ means that X is preferred over Y.

It is necessary to query a sufficient number of pairwise comparisons so that the optimal solution recovers the underlying true scores.[7] It turns out to be sufficient to query a subset of all the possible comparisons as long as each image appears in at least k ($k < N-1$) comparisons presented to the humans, where k can be determined empirically using a very small subset of the data for which human labels are acquired [28]. To decide on k, we collect the exhaustive pairwise comparisons of 6 inter-type groups, where each group contains 15 images. Within each inter-type group, we then analyze ML convergence as a function of k, by using a sampled set of k comparisons per image to solve problem (4). The remaining comparisons are then used to evaluate the estimation accuracy.

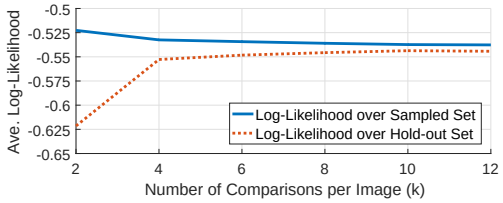


Figure 3: Average log-likelihood as a function of the number of comparisons per image.

Figure 4: Average error rates of ML-estimated binary preference w.r.t. the true preference when $\tilde{p}_{AB,40} \notin [0.35, 0.65]$, as a function of the number of comparisons per image.

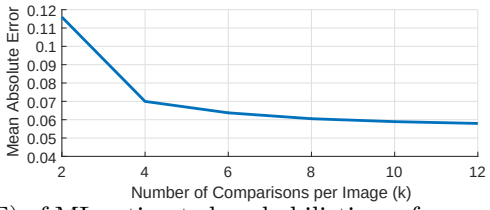


Figure 5: Mean Absolute Error (MAE) of ML-estimated probabilistic preference w.r.t. the true preference, as a function of the number of comparisons per image.

Fig. 3 shows the average log-likelihood over the sampled set and hold-out set, which is the objective function of problem (4) (with a scaling factor). It can be seen that the estimation converges when $k \geq 10$, indicating that no further improvement on estimation accuracy can be obtained beyond $k = 10$. Fig. 4 shows the average error rate of binary preference estimation as a function of k, when $\tilde{p}_{AB,40} \notin [0.35, 0.65]$. When $k = 10$, the binary error rate is 0.0006 over the hold-out set, averaging over different randomizations of the k comparisons per image as well as over the 6 groups of inter-type comparisons. Furthermore, Fig. 5 shows the Mean Absolute Error (MAE) in estimating $\tilde{p}_{AB,40}$. It can be seen that when $k \geq 10$, the estimation MAE is less than 0.06. As such, we have used $k = 10$ in this paper.

Overall, this indicates that we can accurately estimate both the human probabilistic preference and binary preference, using ML estimation, based on a smaller set of human responses. By using ML estimation with $k = 10$, we have reduced the required number of queries by approximately 24%, as compared to exhaustively querying all the possible pairs.[8]

[7] Note that only the score difference carries information. The numeric value of each single score is arbitrary [26].

[8] Note that for each intra group, we query human responses for all possible pairwise comparisons because there are only three images in each group, leading to higher sensitivity of ML estimation to human noise.

5 Limitations of Existing Datasets (Paper Sec. 1)

As discussed in the paper (Sec. 1), existing IQA databases [2, 3, 29–33] typically assign a numerical "quality" score to each individual image, which is referred to as the Mean Opinion Score (MOS). As we extensively discussed, although there may exist an intrinsic quality score for each image, learning it by asking humans to provide subjective ratings to an individual image can be considerably challenging and noise-prone. A few work attempt to address this issue and define a pairwise MOS by collecting scores that capture pairwise comparisons [1,4]. However, the way this pairwise MOS is calculated makes it dependent on the specific set. In other words, the pairwise MOS score of a pair of distorted images can change considerably when the same pair is included in two different datasets. In this supplementary document, we provide a few examples of this, motivating the need for our proposed set-independent ground truth quality label (based on probabilistic pairwise preference). We also demonstrate some of the failure cases in LIVE [2] and CSIQ [3] where the MOS is based on asking humans to assign quality scores to single images directly (the particular technique adopted for doing this involved taking a difference between MOS of a reference and a distorted image, hence lower score is better, and is termed DMOS). To identify such cases, we collect human pairwise probabilistic preference labels for 500 randomly sampled distorted image pairs from both these datasets and identify pairs that are inconsistent with human preference (we select cases where human probabilistic preference is strong, i.e., outside probability range of $[0.35, 0.65]$).

We first demonstrate the set-dependence of MOS calculated in TID2008 and TID2013 datasets (Sec. 5.1), which share several identical images. In Fig. 6-9, we show 4 example image pairs with their pairwise MOS difference calculated based on TID2008 and TID2013, respectively. It can be seen that the pairwise MOS difference can vary considerably from one set (e.g., TID2008) to another set (e.g., TID2013). In some of the examples, this discrepancy of the pairwise MOS difference is so high that it results in a flipped preference, as can be seen. Next we present 3 examples of image pairs for CSIQ (Sec. 5.2, Fig. 11, 12, 13) and LIVE (Sec. 5.3, Fig. 14, 15, 16) datasets in which the binary preference from DMOS labels is inconsistent with human probabilistic preference. One of the factors that could lead to such inconsistencies is the difficulty and subjectivity of the task of assigning a quality score to a single image [1,4].[9]

[9] Although we have only included a few examples of discrepancies here, more such examples can be easily found in the respective datasets.

5.1 Inconsistencies in TID ground-truth quality labeling scheme

reference image

pairwise MOS difference calculated on TID2008 = 0.4667
pairwise MOS difference calculated on TID2013 = -0.6194

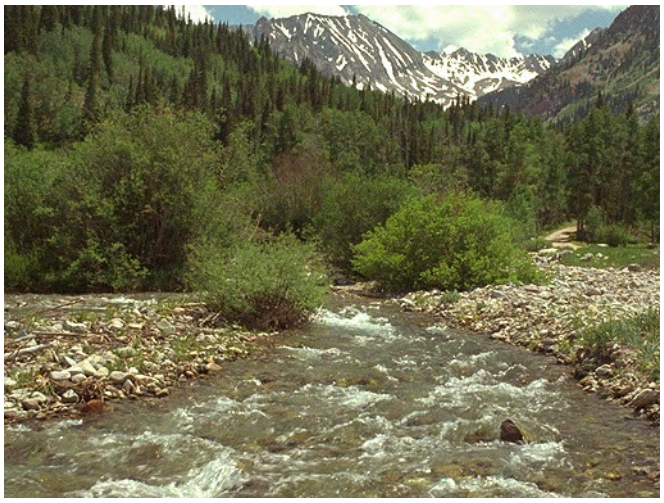
Figure 6: **Limitation of Swiss-tournament-based MOS labeling scheme**: Here is an example of a pair of distorted images with pairwise MOS difference (between MOS of left image and right image) calculated based on TID2008 and TID2013, respectively. The top row shows the undistorted reference image and the bottom row shows the two distorted images. It can be seen that pairwise MOS difference is different depending on the set it is evaluated on. It can further be seen that the pairwise MOS scores indicate a different preference in terms of which image would be preferred by humans, when they are calculated based on TID2008 and TID2013, respectively.

reference image

pairwise MOS difference calculated on TID2008 = -1.0968
pairwise MOS difference calculated on TID2013 = 0.2564

Figure 7: **Limitation of Swiss-tournament-based MOS labeling scheme**: Here is an example of a pair of distorted images with pairwise MOS difference (between MOS of left image and right image) calculated based on TID2008 and TID2013, respectively. The top row shows the undistorted reference image and the bottom row shows the two distorted images. It can be seen that pairwise MOS difference is different depending on the set it is evaluated on. It can further be seen that the pairwise MOS scores indicate a different preference in terms of which image would be preferred by humans, when they are calculated based on TID2008 and TID2013, respectively.

reference image



pairwise MOS difference calculated on TID2008 = -0.4118
pairwise MOS difference calculated on TID2013 = -1.4408

Figure 8: **Limitation of Swiss-tournament-based MOS labeling scheme**: Here is an example of a pair of distorted images with pairwise MOS difference (between MOS of left image and right image) calculated based on TID2008 and TID2013, respectively. The top row shows the undistorted reference image and the bottom row shows the two distorted images. It can be seen that pairwise MOS difference is different depending on the set it is evaluated on.

reference image

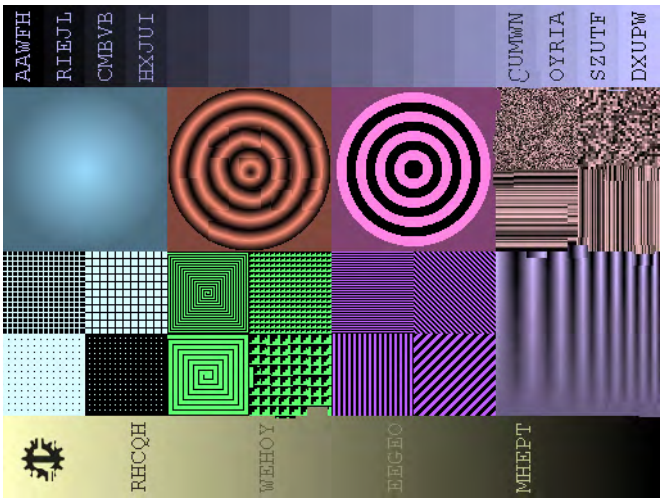
pairwise MOS difference calculated on TID2008 = 1.4839
pairwise MOS difference calculated on TID2013 = 0.2632

Figure 9: **Limitation of Swiss-tournament-based MOS labeling scheme**: Here is an example of a pair of distorted images with pairwise MOS difference (between MOS of left image and right image) calculated based on TID2008 and TID2013, respectively. The top row shows the undistorted reference image and the bottom row shows the two distorted images. It can be seen that pairwise MOS difference is different depending on the set it is evaluated on.

TID2008 MOS: 6.5143
TID2013 MOS: 5.1351

TID2008 MOS: 7.2941
TID2013 MOS: 5.9189

TID2008 MOS: 3.9722
TID2013 MOS: 2.9714

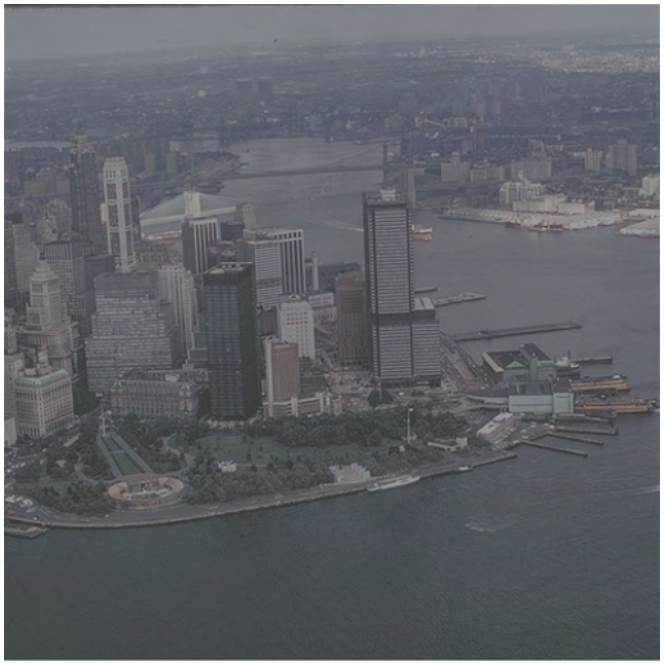


TID2013 MOS: 4.0769
TID2013 MOS: 5.0882

Figure 10: **Limitation of Swiss-tournament-based MOS labeling scheme**: Example images with individual MOS reported in TID2008 and TID2013, respectively. It can be seen that the numerical values of MOS of an exactly same image (L2 error = 0) in the two databases can change drastically when obtained in two different sets.

5.2 Inconsistencies in CSIQ ground-truth quality labeling scheme

reference image



DMOS score difference calculated on CSIQ = -0.01
human probability of preferring left image over right based on Amazon Mechanical Turk responses = 0.06

Figure 11: **Inconsistency of CSIQ DMOS labeling scheme**: Here is an example of a pair of distorted images with pairwise DMOS difference (between DMOS of left image and right image; with individual DMOS ranging from 0 to 1 and lower score indicating a better image) calculated based on CSIQ dataset reported in the top row. The human probability of preferring left image over right is reported at the bottom. A negative value for DMOS difference indicates that left image is preferred over the right one as per CSIQ DMOS label. However, humans prefer the right image over left.

reference image

DMOS score difference calculated on CSIQ = -0.04
human probability of preferring left image over right based on Amazon Mechanical Turk responses = 0.18

Figure 12: **Inconsistency of CSIQ DMOS labeling scheme**: Here is an example of a pair of distorted images with pairwise DMOS difference (between DMOS of left image and right image; with individual DMOS ranging from 0 to 1 and lower score indicating a better image) calculated based on CSIQ dataset reported in the top row. The human probability of preferring left image over right is reported at the bottom. A negative value for DMOS difference indicates that left image is preferred over the right one as per CSIQ DMOS label. However, humans prefer the right image over left.

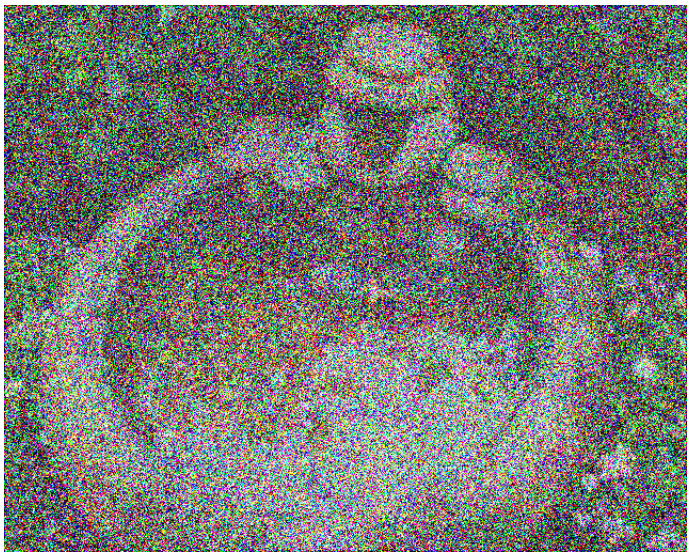
reference image

MOS score difference calculated on CSIQ = -0.05
human probability of preferring left image over right based on Amazon Mechanical Turk responses = 0.20

Figure 13: **Inconsistency of CSIQ DMOS labeling scheme**: Here is an example of a pair of distorted images with pairwise DMOS difference (between DMOS of left image and right image; with individual DMOS ranging from 0 to 1 and lower score indicating a better image) calculated based on CSIQ dataset reported in the top row. The human probability of preferring left image over right is reported at the bottom. A negative value for DMOS difference indicates that left image is preferred over the right one as per CSIQ DMOS label. However, humans prefer the right image over left.

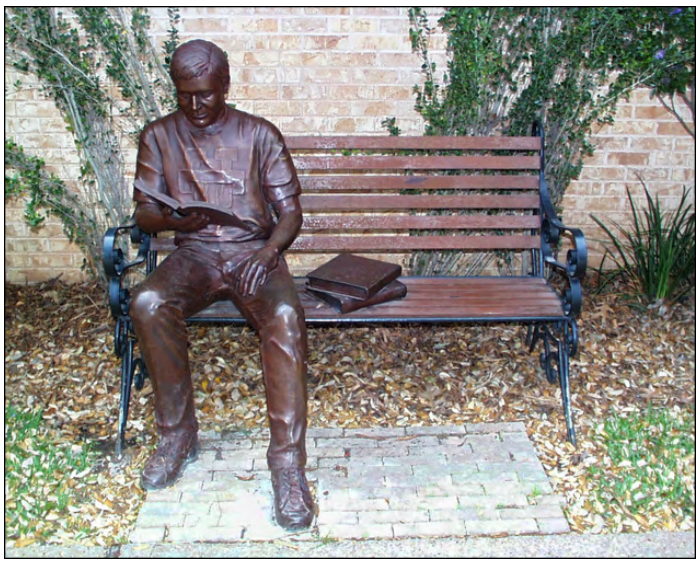
5.3 Inconsistencies in LIVE ground-truth quality labeling scheme

reference image



DMOS score difference calculated on LIVE = -0.93
human probability of preferring left image over right based on Amazon Mechanical Turk responses = 0.01

Figure 14: **Inconsistency of LIVE DMOS labeling scheme**: Here is an example of a pair of distorted images with pairwise DMOS difference (between MOS of left image and right image; with individual DMOS ranging from 0 to 100 and lower score indicating a better image) calculated based on LIVE dataset reported in the top row. The human probability of preferring left image over right is reported at the bottom. A negative value for DMOS difference indicates that left image is preferred over the right one as per LIVE DMOS label. However, humans prefer the right image over left.



reference image

DMOS score difference calculated on LIVE = -7.34
human probability of preferring left image over right based on Amazon Mechanical Turk responses = 0.20

Figure 15: **Inconsistency of LIVE DMOS labeling scheme**: Here is an example of a pair of distorted images with pairwise DMOS difference (between MOS of left image and right image; with individual DMOS ranging from 0 to 100 and lower score indicating a better image) calculated based on LIVE dataset reported in the top row. The human probability of preferring left image over right is reported at the bottom. A negative value for DMOS difference indicates that left image is preferred over the right one as per LIVE DMOS label. However, humans prefer the right image over left.

reference image

DMOS score difference calculated on LIVE = -6.2
human probability of preferring left image over right based on Amazon Mechanical Turk responses = 0.10

Figure 16: **Inconsistency of LIVE DMOS labeling scheme**: Here is an example of a pair of distorted images with pairwise DMOS difference (between MOS of left image and right image; with individual DMOS ranging from 0 to 100 and lower score indicating a better image) calculated based on LIVE dataset reported in the top row. The human probability of preferring left image over right is reported at the bottom. A negative value for DMOS difference indicates that left image is preferred over the right one as per LIVE DMOS label. However, humans prefer the right image over left.

6 Details of the Image Distortions (Paper Sec. 4)

In this section, we describe the implementation details of the training and test distortions in our proposed database, as well as the reason why each distortion is included in the database (see the note on "Relevance" for each distortion). In the implementation descriptions, we have included information on how we select the distortion parameters. Unless specified otherwise, the image pixel values range from **0** to **1**. For each distortion, 2-4 pictorial examples are also provided, which show a few different distortion levels.

The different levels of a given distortion are generated by varying one or more parameters involved (e.g., the blur kernel variance for Gaussian blur). In the following descriptions, we specify the parameter ranges used for each distortion in our database. These parameter ranges are empirically chosen such that the distorted images do not exhibit drastic/unrealistic effects upon manual visual assessment. Each distorted image is generated by applying a distortion to the reference image, with randomly-sampled parameters. The reference image used is shown in Fig. 17. In the following descriptions, we denote the distorted image by $I_D(x, y, c)$ and reference image by $I_R(x, y, c)$, where (x, y) denotes the pixel location and c denotes a channel.

As mentioned in the paper, the distortions are grouped according to their visual effects. The descriptions of the distortions in each group can be easily accessed via the hyperlinks in Table 3 and Table 4.

Note that 16 of the distortions in our proposed training set have also been considered in the largest existing IQA database [4]. These 16 distortions are additive Gaussian noise with higher noise levels in color channels, multiplicative Gaussian noise, spatially correlated noise, masked noise, salt and pepper noise (impulse noise), JPEG compression, local blockwise color distortion, non-eccentricity pattern noise, JPEG2000 transmission errors, lossy compression of images corrupted by additive Gaussian noise, Gaussian blur, change of saturation, color quantization, color quantization with dither, mean color shift, and chromatic aberrations.

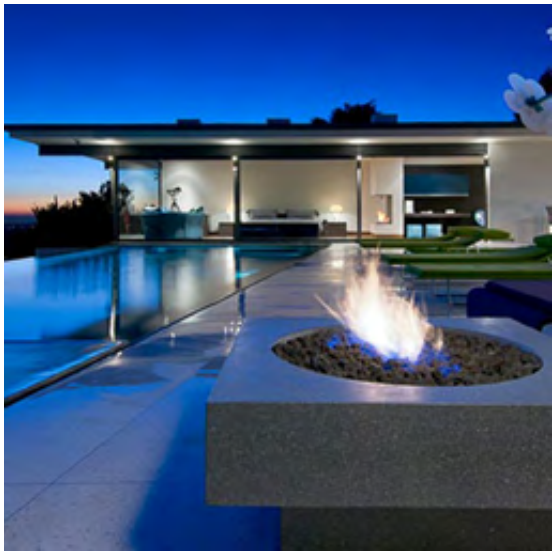


Figure 17: undistorted reference image

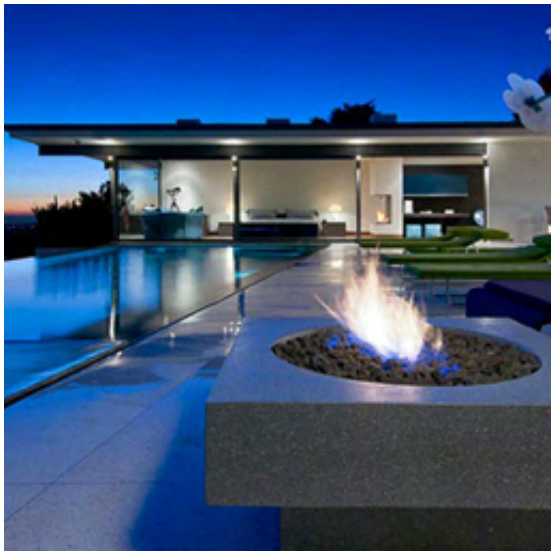
6.1 Training image distortions

Visual Effects	Training Image Distortions
Noise	1. additive Gaussian noise with higher noise levels in color channels 2. multiplicative Gaussian noise 3. spatially-correlated noise 4. masked noise 5. salt and pepper noise
Block-like artifacts	6. JPEG compression 7. zeroing out frequency components from all image blocks 8. zeroing out frequency components from randomly selected image blocks 9. local blockwise color distortion 10. non-eccentricity pattern noise 11. block-based image hole-filling (using a Poisson solver)
Artifacts with regular patterns	12. deblurring using Lucy-Richardson's method [34, 35] for images corrupted by Gaussian blur 13. deblurring using Lucy-Richardson's method for images corrupted by motion blur 14. joint deblurring and denoising using Lucy-Richardson's method for images corrupted by additive Gaussian noise and Gaussian blur 15. joint deblurring and denoising using Lucy-Richardson's method for images corrupted by additive Gaussian noise and motion blur 16. JPEG2000 transmission errors 17. lossy compression of images corrupted by additive Gaussian noise 18. zeroing out frequency components from an image
Detail loss	19. Gaussian blur 20. motion blur 21. locally-varying blur 22. Perona-Malik denoising [36] 23. Rudin-Osher-Fatemi (ROF) denoising [37] 24. median denoising of image corrupted by salt and pepper noise 25. deep network-based super-resolution with a sparse prior [38]
Color change	26. change of saturation 27. contrast stretching 28. gamma transformation 29. local color shift 30. local contrast change 31. color quantization 32. color quantization with dither 33. mean color shift
Geometric transformations	34. projective transformation 35. warping (using a local spatial shift map) 36. spatial shift 37. 2D rotation 38. spatial shift and rotation 39. radial barrel transform 40. radial pincushion transform
Others	41. compressive sensing using Orthogonal Matching Pursuit [39] 42. chromatic aberrations 43. JPEG transmission error 44. image sharpening

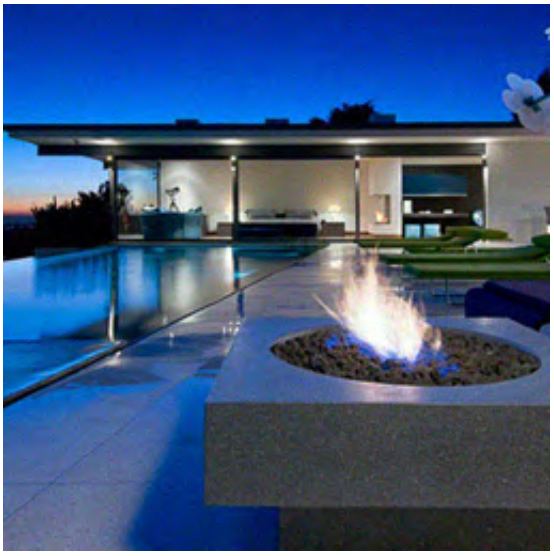
Table 3: Training distortions are categorized according to their visual effects. Implementation details and pictorial examples for each distortion can be found by following the hyperlink of each visual effect category.

6.1.1 Noise

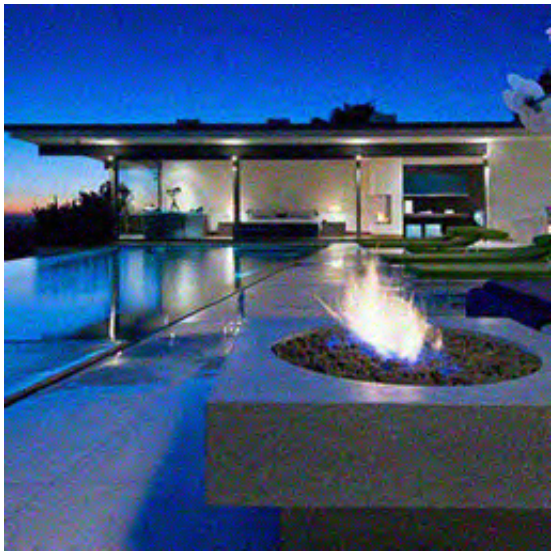
1. Additive Gaussian noise with higher noise in color channels: Additive Gaussian noise is applied to each of the luminance and chrominance channels of an image (represented using YCbCr color space). The noise variance of the chrominance channels, σ_b^2 and σ_r^2, are higher than that of the luminance channel (σ_l^2). More specifically, $\sigma_b = \alpha_b \times \sigma_l$ and $\sigma_r = \alpha_r \times \sigma_l$, where $\alpha_b \geq 1$ and $\alpha_r \geq 1$.
Parameters: Noise standard deviation in the luminance channel: $\sigma_l \in [0.005, 0.03]$ and the scaling factors $\alpha_b, \alpha_r \in [1, 2.8]$
Relevance: This distortion is used to capture the known property that the Human Visual System (HVS) do not equally perceive distortions in the brightness (luminance) and the color (chrominance) components [1,4].



(a) $\sigma_l = 0.005,\ \alpha_b = 1.2,\ \alpha_r = 1$



(b) $\sigma_l = 0.01,\ \alpha_b = 1.8,\ \alpha_r = 1.6$



(c) $\sigma_l = 0.02,\ \alpha_b = 2.7,\ \alpha_r = 2.2$

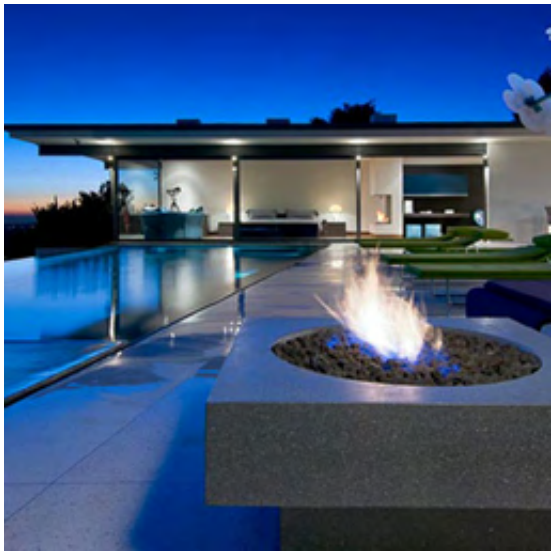
(d) $\sigma_l = 0.03,\ \alpha_b = 2.8,\ \alpha_r = 2.8$

Figure 18: This figure shows 4 sample images corrupted by additive Gaussian noise with higher noise variances in color channels. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

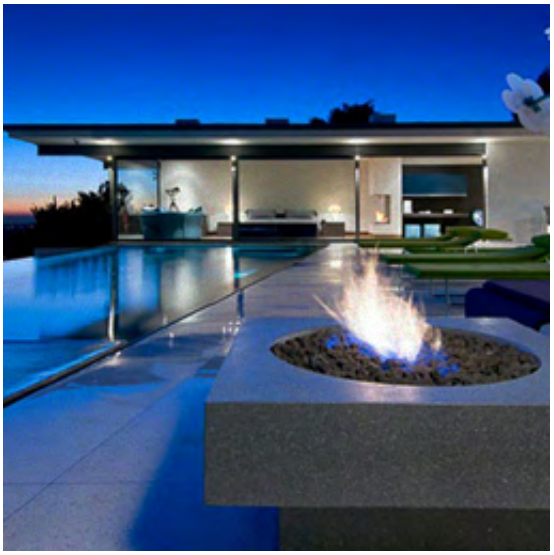
2. Multiplicative Gaussian noise: The distorted image $I_D(x, y, c)$ is computed using the following formula:

$$I_D(x, y, c) = I_R(x, y, c) \times N(x, y, c) + I_R(x, y, c) = I_R(x, y, c) \times (N(x, y, c) + 1), \quad (5)$$

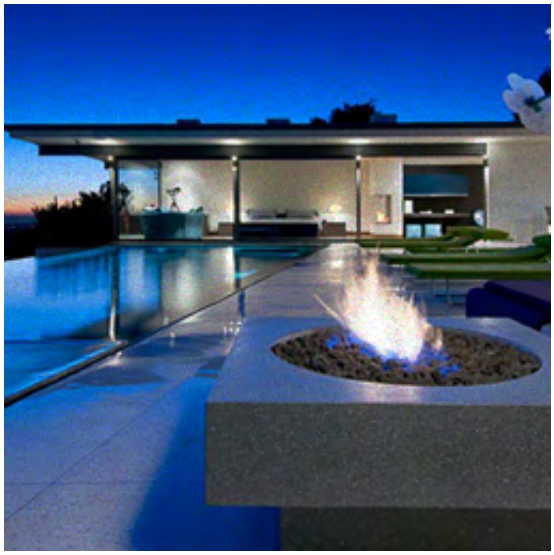
where $I_R(x, y, c)$ is the reference image and $N(x, y, c) \sim \mathcal{N}(0, \sigma_g^2)$.
Parameter: Gaussian noise standard deviation: $\sigma_g \in [0.01, 0.08]$
Relevance: This distortion can be caused by the image acquisition process.



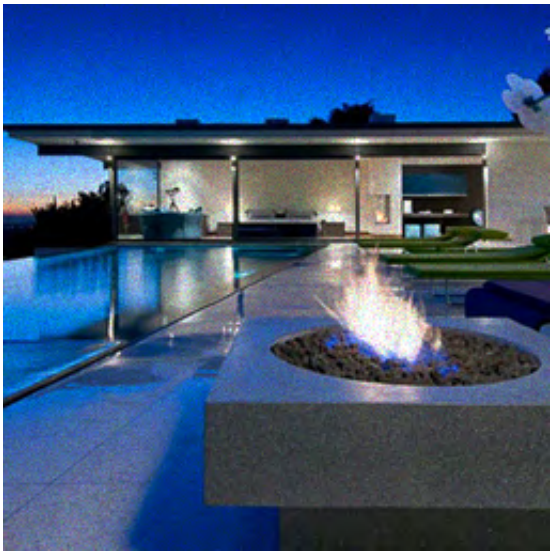
(a) $\sigma_g = 0.01$



(b) $\sigma_g = 0.03$



(c) $\sigma_g = 0.05$



(d) $\sigma_g = 0.08$

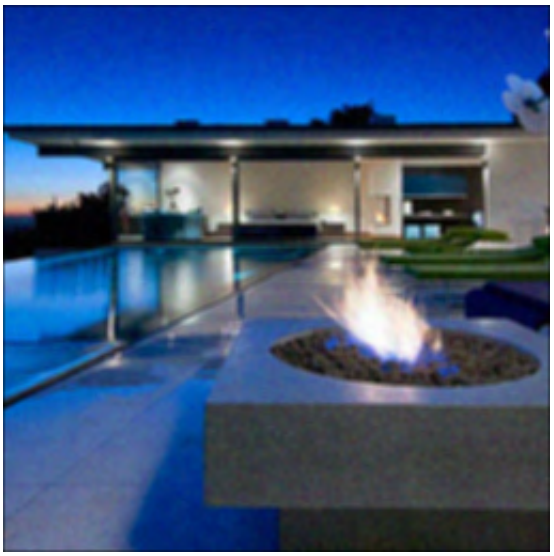
Figure 19: This figure shows 4 sample images corrupted by multiplicative Gaussian noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

3. Spatially-correlated noise: The reference image is first corrupted by an additive Gaussian noise, which results in each pixel being corrupted by an independent and identically distributed noise pattern. The resultant image is then filtered with an average filter of kernel size 3×3, correlating the intensity of each pixel with those of the neighboring pixels. More specifically, the distorted image is given by:

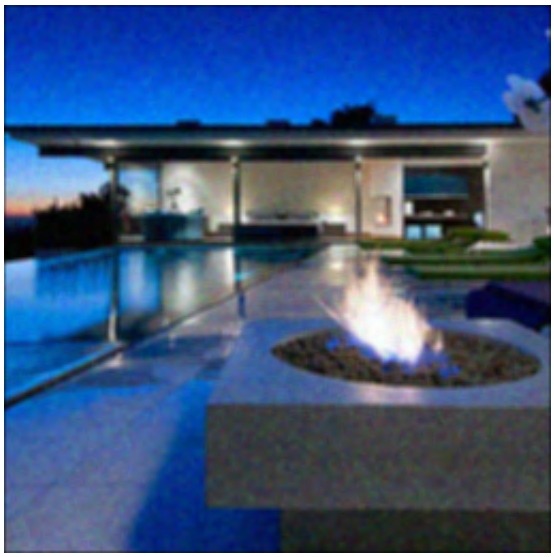
$$I_D(x, y, c) = \frac{1}{|N_n|} \sum_{i \in N_n} (I_R(x_i, y_i, c_i) + N(x_i, y_i, c_i)), \tag{6}$$

where $I_D(x, y, c)$ is the distorted image, $I_R(x, y, c)$ is the reference image, N_n is the set of neighboring pixels, and $N(x, y, c) \sim \mathcal{N}(0, \sigma_g^2)$.
Parameter: Additive Gaussian noise variance: $\sigma_g^2 \in [10^{-5}, 4 \times 10^{-3}]$
Relevance: Additive white noise can get spatially correlated at various stages in a camera image pipeline (e.g., demosaicing).

(a) $\sigma_g = 10^{-5}$



(b) $\sigma_g = 0.03$



(c) $\sigma_g = 0.0025$

(d) $\sigma_g = 0.004$

Figure 20: This figure shows 4 sample images corrupted by spatially-correlated noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

4. Masked noise : There are two implementations of masked noise in our database (both computed in a per-channel manner):

1. *Using Contrast Sensitivity Function (CSF)-based masking:* For this distortion, the visibility of Gaussian high-frequency noise (Sec. 6.2.1 of this file, test distortion No. 2) in an image region is modulated by the masking effect (computed based on the CSF) of that region [1, 40]. The distorted image $I_D(x, y)$ is computed as:

$$I_D(x, y) = M(x, y) \circ I_H(x, y) + (1 - M(x, y)) \circ I_R(x, y), \tag{7}$$

where $I_H(x, y)$ is an image corrupted by Gaussian high frequency noise with frequency selection threshold set to 0.5, $M(x, y)$ is the mask map, and "$\circ$" denotes the Hadamard product of two matrices.

Parameter: Standard deviation of Gaussian high frequency noise: $\sigma_g \in [40, 120]$. (The pixel values range from 0 to 255 in this case.)

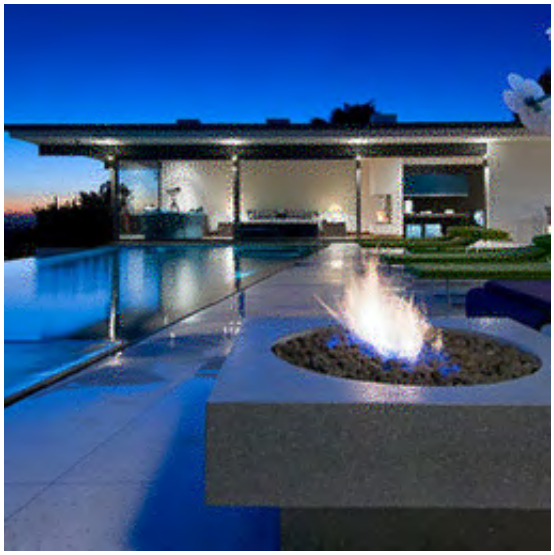
2. *Using Watson's DCT-based visual model:* Watson's DCT-based visual model utilizes luminance and contrast masking while computing *slacks*, s_l (a measure of the amount by which a DCT coefficient can be altered without causing noticeable artifacts), for each component of all 8×8 block-DCT coefficients of an image [41]. A signal (a vector, $\mathbf{m}$, which is drawn from a Gaussian distribution of variance 0.1 in our case) is then added to an original DCT coefficient, C_0, (during the process of digital watermarking, for example) resulting in C_w as follows:

$$C_w(i, j, k) = C_0(i, j, k) + \alpha_m \times m \times s_l(i, j, k), \tag{8}$$

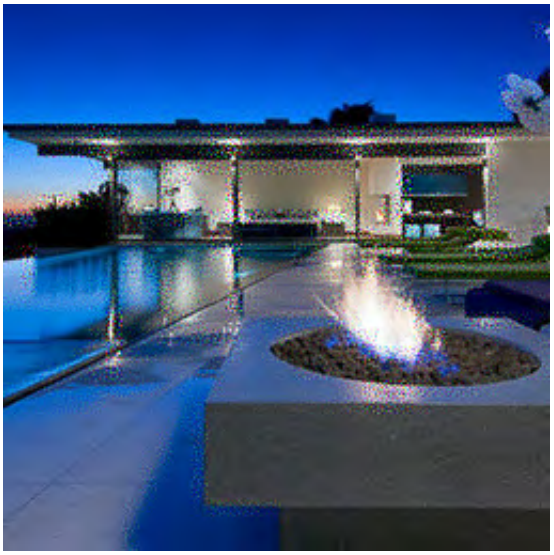
where α_m is a scaling factor, the indices (i, j, k) represent $(i, j)^{th}$ value of k^{th} 8×8 block of a 2D matrix and m is an element in $\mathbf{m}$. A popular practice for DCT-based watermarking is to utilize mid-band frequencies for proper concealment and robustness to compression [42, 43]. We choose the mid-band DCT coefficients at positions $8, 9, 12, 13, 14, 17, 18, 19, 20, 23, 24, 25, 26, 27$ (in the zig-zag scan order used for JPEG compression) for concealing $\mathbf{m}$. The coefficients C_w are then used to construct the distorted image. To simulate imperfect concealment, we vary α_m.

Parameter: $\alpha_m \in [0.8, 2.8]$.

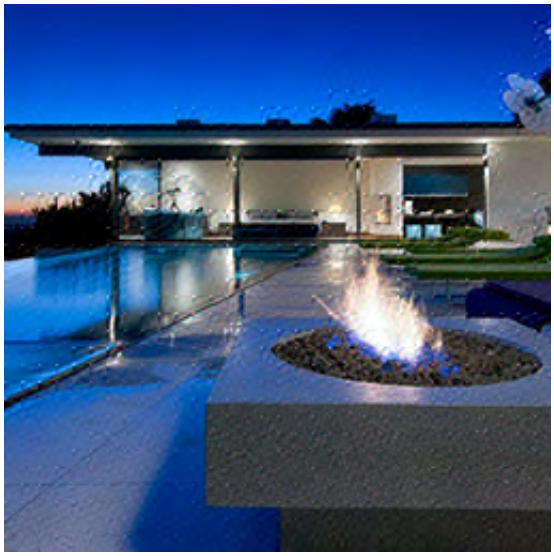
Relevance: The concept of masking is used in digital watermarking [41] and captures the sensitivity of the HVS to contrast and luminance changes.



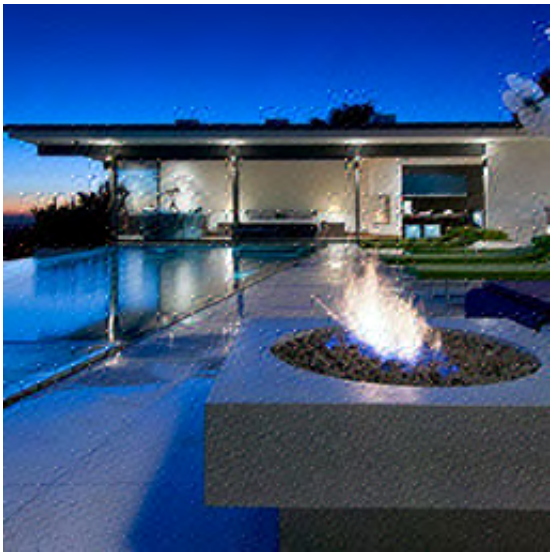
(a) CSF-based, $\sigma_g = 60$



(b) CSF-based, $\sigma_g = 120$



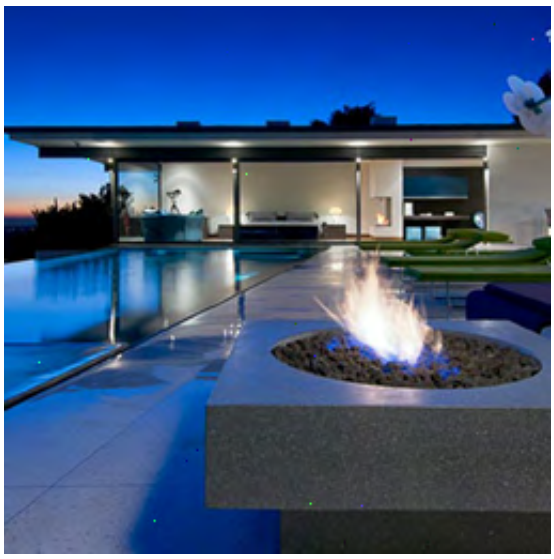
(c) Watson's model-based, $\alpha_m = 0.8$



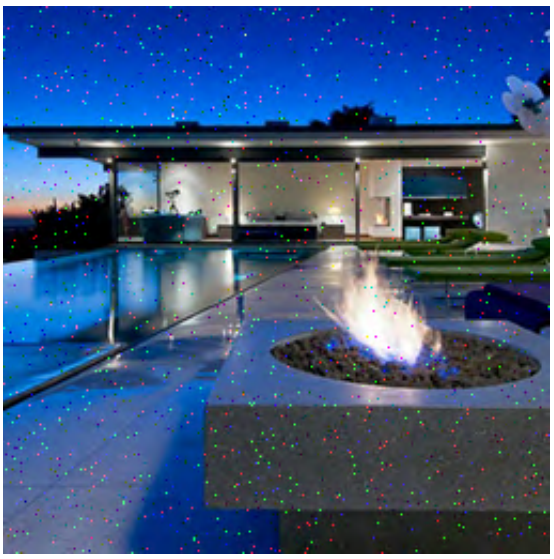
(d) Watson's model-based, $\alpha_m = 1.0$

Figure 21: This figure shows 4 sample images corrupted by masked noise. The top row shows the cases with CSF-based masked noise and the bottom row shows the cases with Watson's model-based masked noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

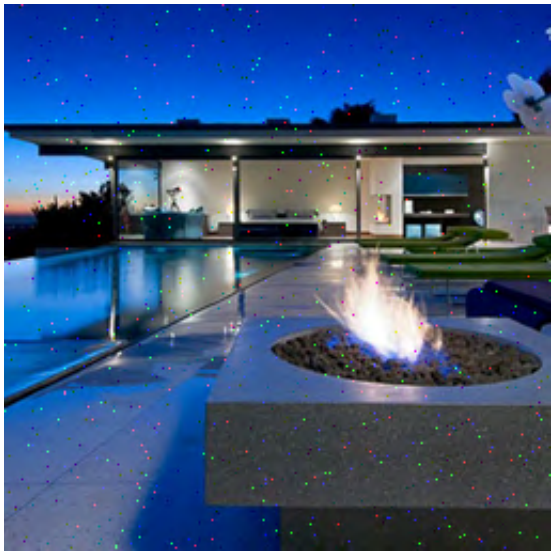
5. Salt and pepper noise: A distorted image is generated by adding salt and pepper noise to the reference image (using the *imnoise* function in Matlab).
Parameter: Density of the noise: $d \in [10^{-4}, 0.045]$
Relevance: This noise can occur during image acquisition, due to errors in the camera imaging pipeline.



(a) $d = 10^{-4}$



(b) $d = 0.01$



(c) $d = 0.005$

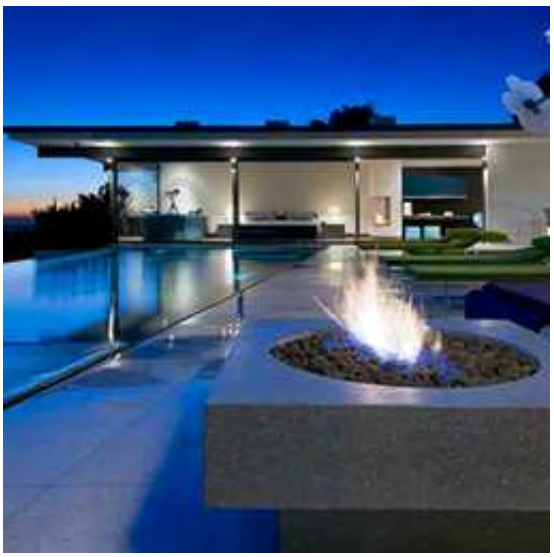
(d) $d = 0.045$

Figure 22: This figure shows 4 sample images corrupted by salt and pepper noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

6.1.2 Block-like artifacts

6. JPEG compression: The distorted image is a JPEG-compressed version of the reference image. JPEG compression is implemented using Matlab's *imwrite* function.
Parameter: Compression quality factor: $q \in \{10, 11, ..., 79, 80\}$
Relevance: This distortion is commonly caused by image compression operations.

(a) $q = 80$



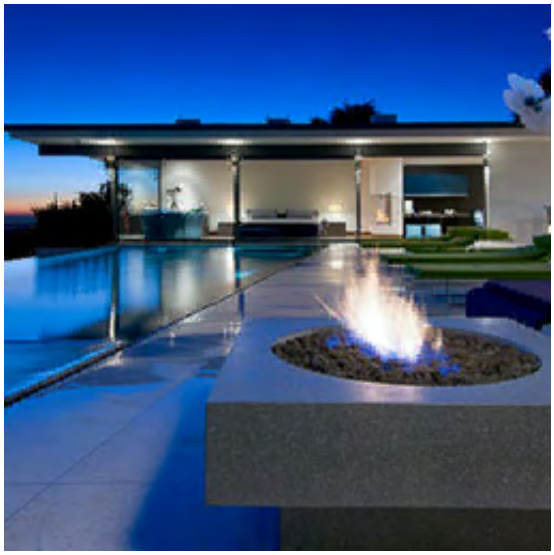
(b) $q = 50$

(c) $q = 30$

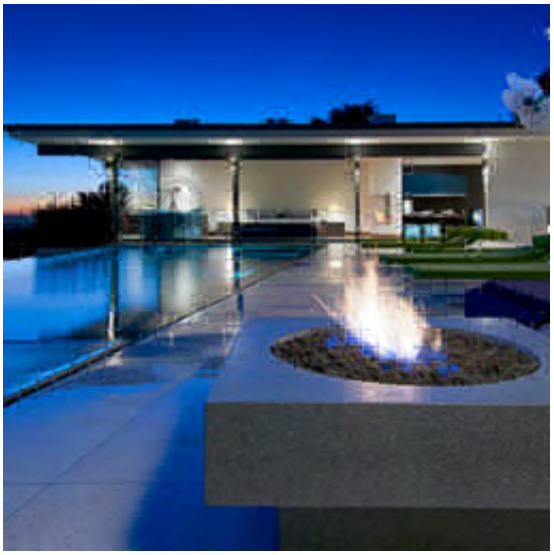
(d) $q = 20$

Figure 23: This figure shows 4 sample images corrupted by JPEG compression artifacts. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

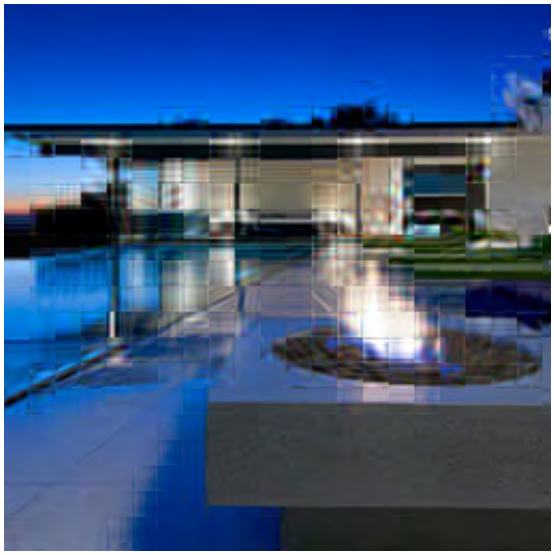
7. Zeroing out frequency components from all image blocks: The input reference image is first divided into blocks. For each block, the frequency components (obtained after applying Fourier transform to the block) whose magnitude falls below a threshold are zeroed out. The threshold is specified in terms of a percentile value (e.g., all frequency components with magnitudes smaller than the 98^{th} percentile are removed from Fig. 24d).
Parameters: The block size: $s_b \in \{4, 5, ..., 20\}$ and the percentile value for computing the magnitude threshold: $per \in [60, 98]$
Relevance: This distortion is used to simulate image compression artifacts.



(a) $s_b = 4,\ per = 60$



(b) $s_b = 8,\ per = 80$

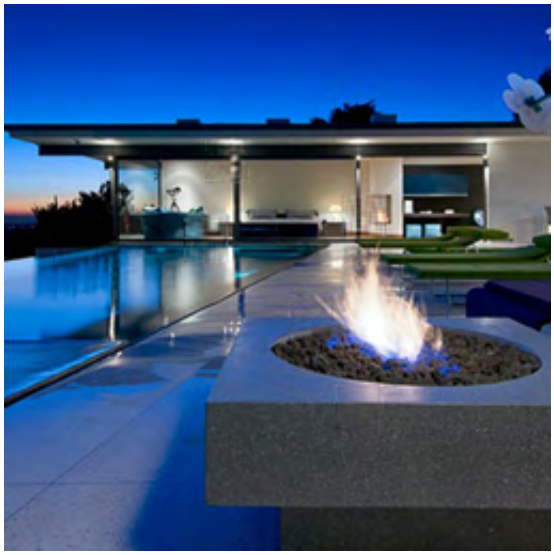


(c) $s_b = 12,\ per = 93$

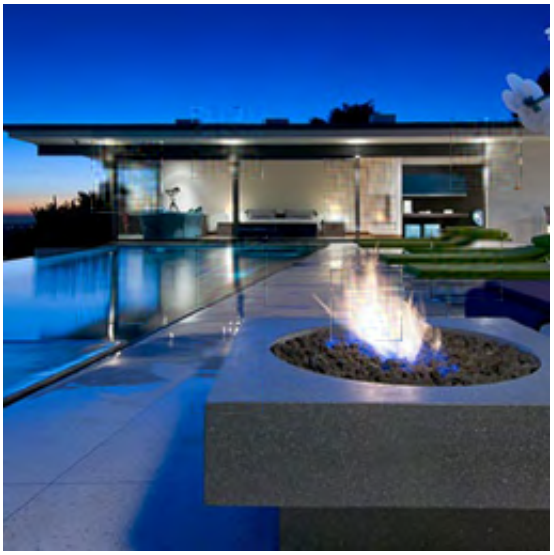
(d) $s_b = 20,\ per = 98$

Figure 24: This figure shows 4 sample images corrupted by the distortion involving zeroing out frequency components from all image blocks. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

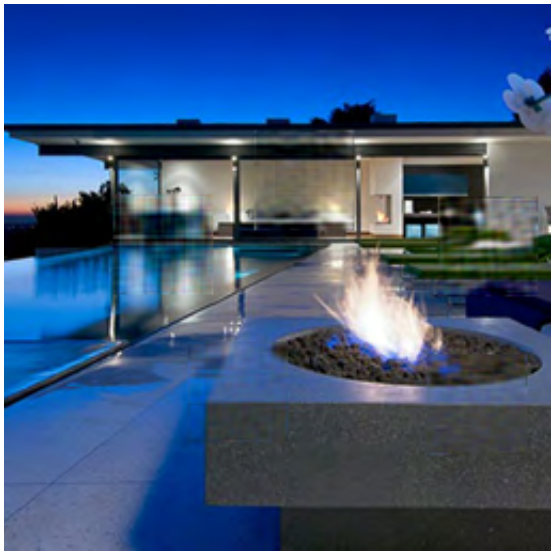
8. Zeroing out frequency components from randomly-selected image blocks: Given a reference image, random image blocks are selected. The frequency components within each selected block (obtained after applying Fourier transform to each block) whose magnitude falls below a threshold are zeroed out. The threshold (same for all selected blocks) is specified in terms of a percentile value (e.g., all frequency components with magnitudes smaller than the 98^{th} percentile are removed from Fig. 25d).
Parameters: The number of random blocks: $n_b \in \{10, 11, ..., 30\}$, the block size: $s_b \in \{4, 5, ..., 40\}$, and the percentile value for computing the magnitude threshold: $per \in [60, 98]$
Relevance: This distortion is used to simulate image compression artifacts.



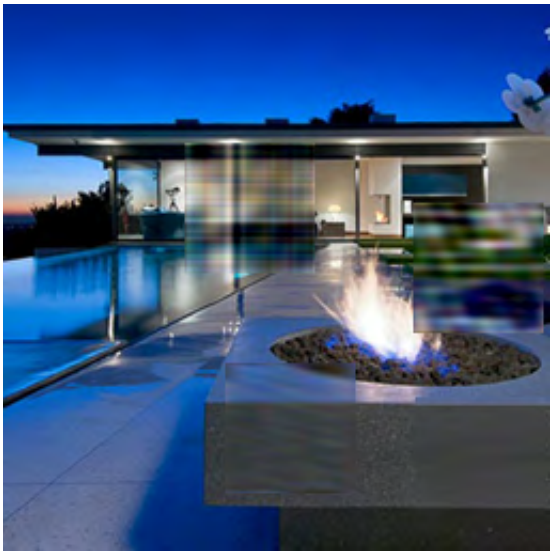
(a) $n_b = 10,\ s_b = 20,\ per = 80$



(b) $n_b = 10,\ s_b = 33,\ per = 86$



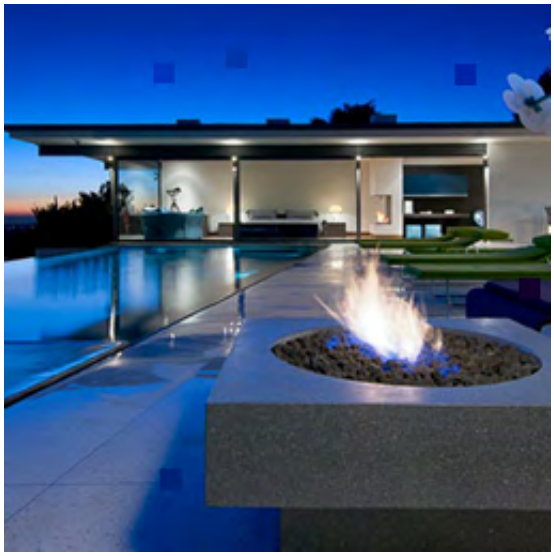
(c) $n_b = 20,\ s_b = 46,\ per = 92$



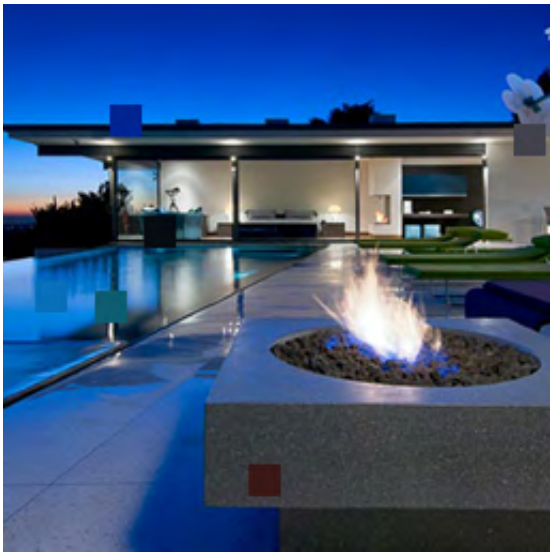
(d) $n_b = 30,\ s_b = 60,\ per = 98$

Figure 25: This figure shows 4 sample images corrupted by the distortion involving zeroing out frequency components from randomly-selected image blocks. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

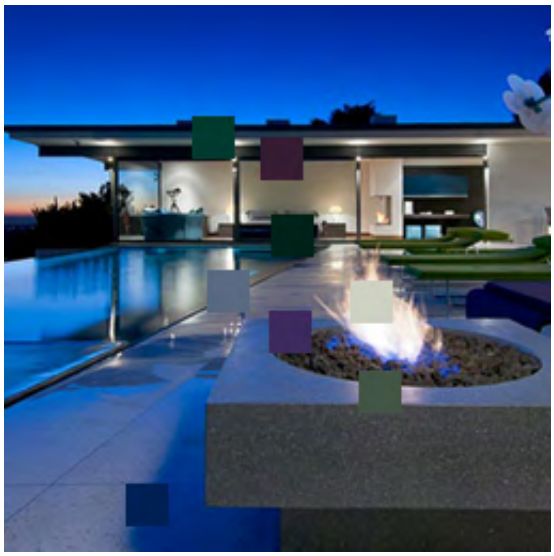
9. Local blockwise color distortion: This distortion is modeled as an arbitrary color shift (from the mean color) in local blocks. The blocks where the color is changed are either selected randomly or based on a saliency map [44]. For saliency map-based selection, all image locations that are one standard deviation above the mean saliency value are isolated. For each image block whose pixel center is contained in these regions, the pixel intensities are first replaced by the block mean and then shifted by a certain amount (independently done for each channel). For randomly-selected blocks, the same color change operation is performed.
Parameters: The number of blocks with such a color change: $n_b \in [4, 10]$, the size of these blocks: $s_b \in [5, 21]$, and the width of the zero-mean uniform distribution from which the color shift for an image block is drawn: $w \in [0, 0.6]$
Relevance: This distortion is used to simulate image inpainting and the image acquisition process [1, 4].



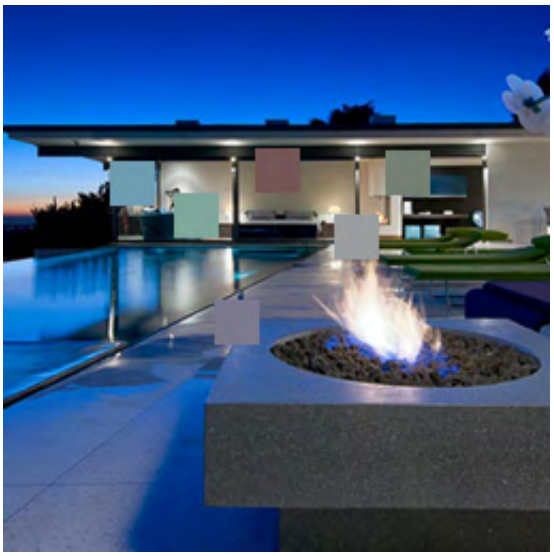
(a) $n_b = 5,\ s_b = 10,\ w = 0.3$



(b) $n_b = 6,\ s_b = 15,\ w = 0.4$



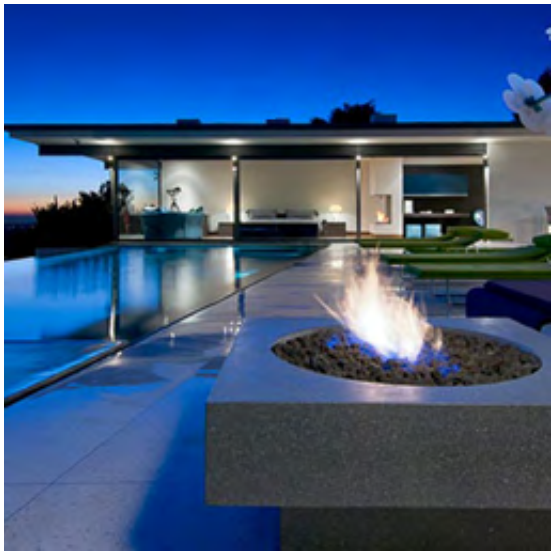
(c) $n_b = 8,\ s_b = 20,\ w = 0.6$



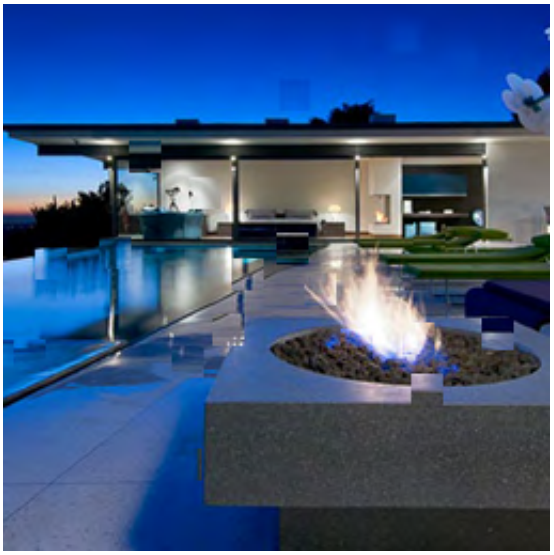
(d) $n_b = 6,\ s_b = 21,\ w = 0.6$

Figure 26: This figure shows 4 sample images corrupted by the local blockwise color distortion. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

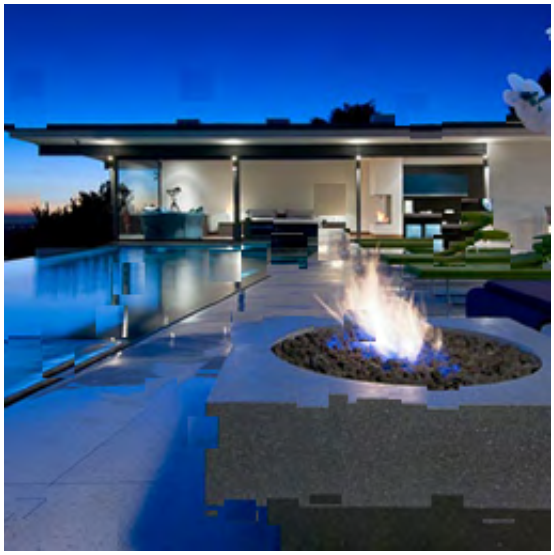
10. Non-eccentricity pattern noise: Given a reference image, the distorted image is obtained by replacing some randomly-selected image patches with randomly-selected neighboring patches (located within a distance of 15 pixels between patch centers). The patch size is fixed to be 15×15.
Parameter: The number of patches to be replaced: $n_p \in \{2, 3, ..., 75\}$
Relevance: Spatial shifts in local image blocks can often go unnoticed by the HVS depending on the conspicuousness of the blocks [1, 4]. This characteristic of the HVS can play a key role in various algorithms related to image synthesis and image compression.



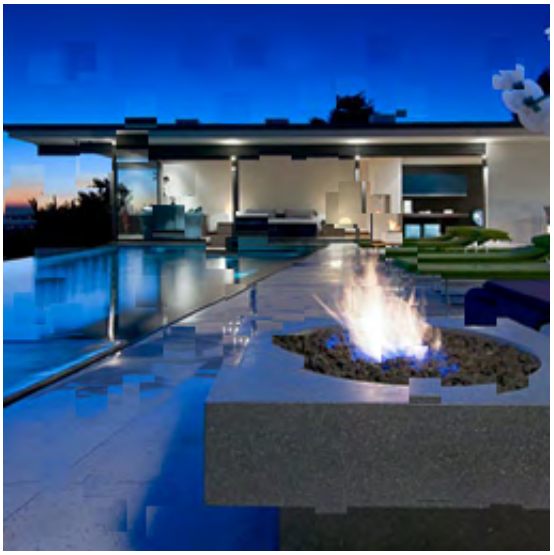
(a) $n_p = 2$



(b) $n_p = 22$



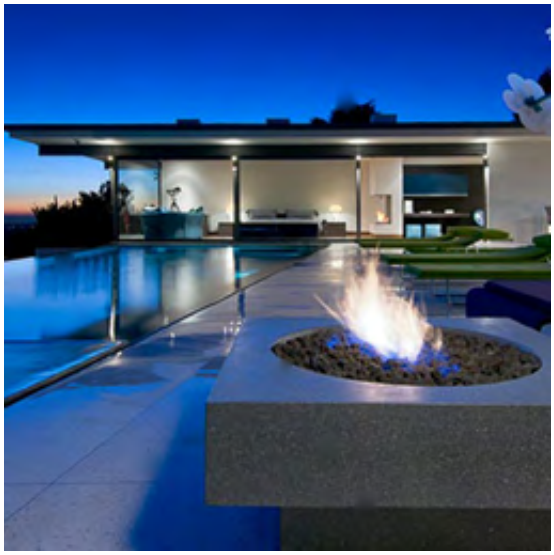
(c) $n_p = 47$



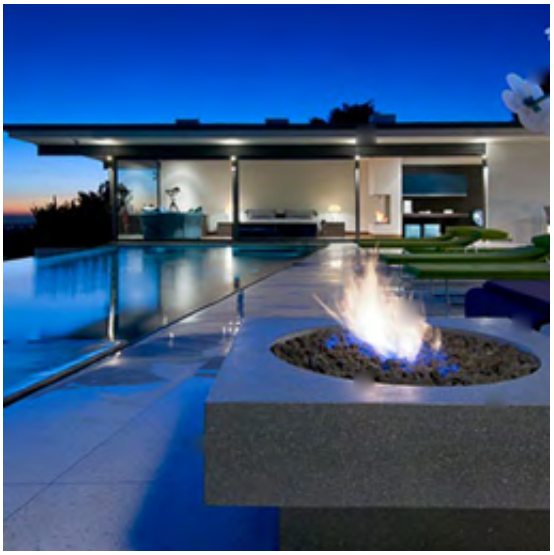
(d) $n_p = 72$

Figure 27: This figure shows 4 sample images corrupted by the non-eccentricity pattern noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

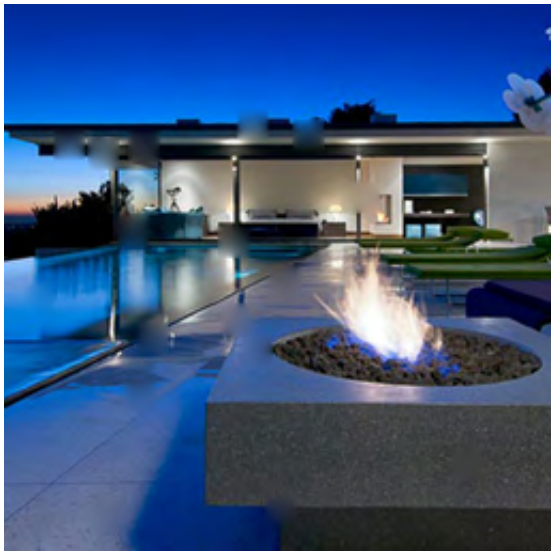
11. Block-based image hole-filling (using a Poisson solver): Given a reference image, randomly-selected square blocks of the image are considered as *holes*. The holes are filled using the *regionfill* function in Matlab, which computes a Laplacian over the holes to be filled and solves a Dirichlet boundary value problem.
Parameters: The number of blocks (holes): $n_b \in \{10, 11, ..., 30\}$ and the width of a block: $s_b \in \{10, 11, ..., 20\}$
Relevance: This distortion is used to simulate the artifacts from the image inpainting and the image synthesis processes.



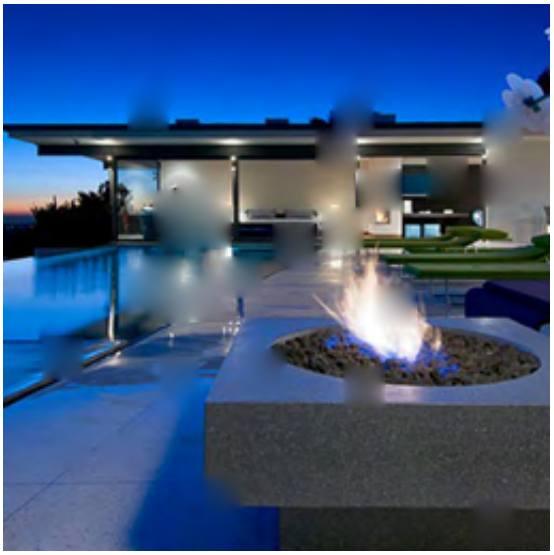
(a) $n_b = 10,\ s_b = 10$



(b) $n_b = 15,\ s_b = 10$



(c) $n_b = 20,\ s_b = 15$

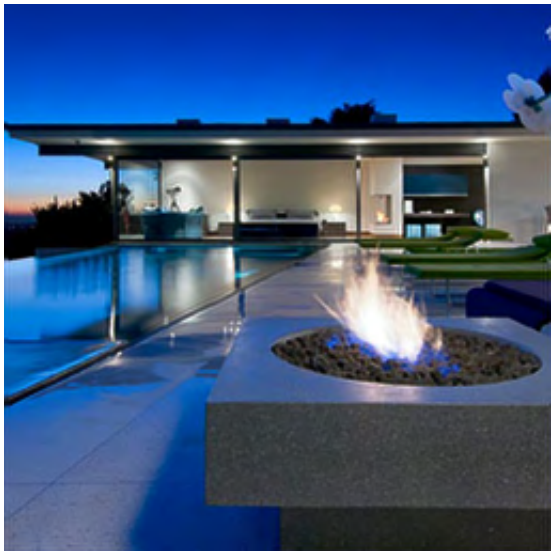


(d) $n_b = 30,\ s_b = 20$

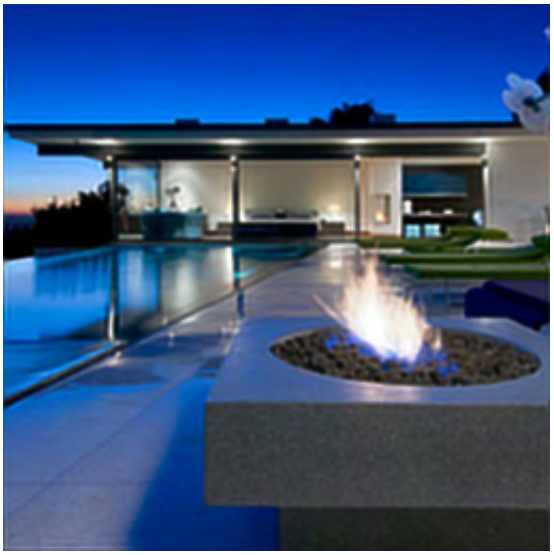
Figure 28: This figure shows 4 sample distorted images with block-based image hole-filling. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

6.1.3 Artifacts with regular patterns

12-13. Deblurring using Lucy-Richardson's method [34, 35]: The input reference image is first blurred (Gaussian blur or linear motion blur). We then deblur the image using Lucy-Richardson's method (using the *deconvlucy* function in Matlab). For Gaussian blur, we set the kernel size (s_k) to be a function of the standard deviation (σ_k): $s_k = 3 \times \sigma_k + 2$.
Parameters: Gaussian blur kernel standard deviation: $\sigma_k \in [0.5, 2.5]$, motion blur kernel length: $len \in [3, 10]$, and the motion blur kernel direction: $dir \in [0, 360]$
Relevance: These distortions are representative of the artifacts caused by a number of image deblurring algorithms.



(a) $\sigma_k = 0.5$

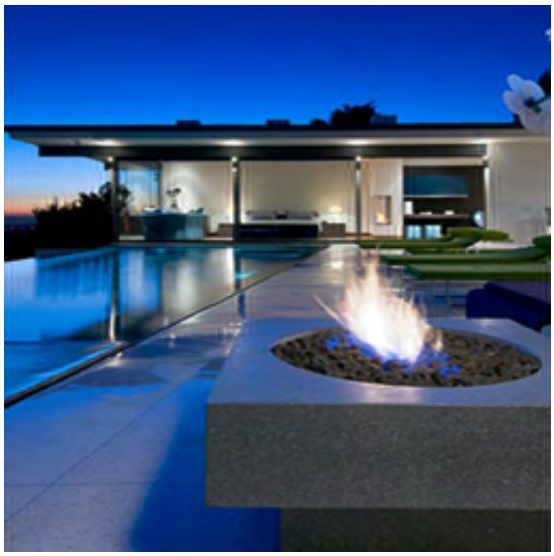


(b) $\sigma_k = 1$

(c) $\sigma_k = 1.5$

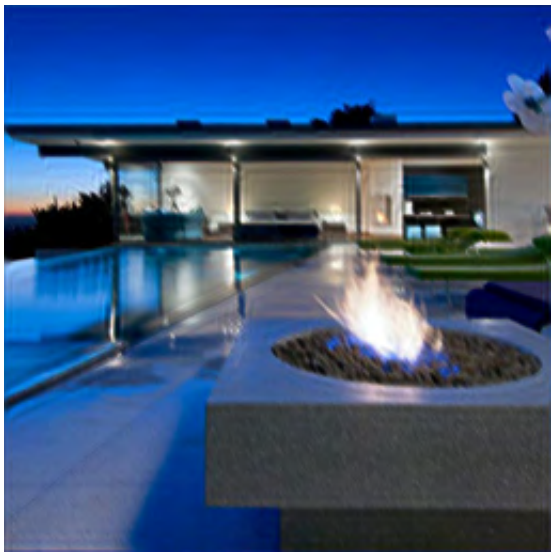
(d) $\sigma_k = 2.5$

Figure 29: This figure shows 4 sample images obtained by applying the deblurring operation using Lucy-Richardson's method [34, 35] to images corrupted by Gaussian blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

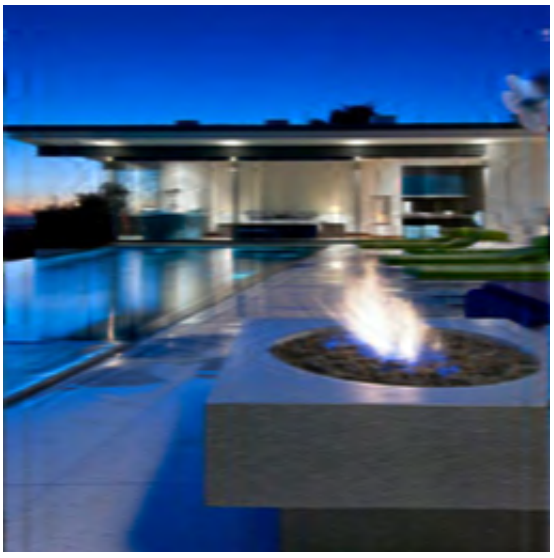


(a) $len = 3,\ dir = 0$

(b) $len = 7,\ dir = 240$



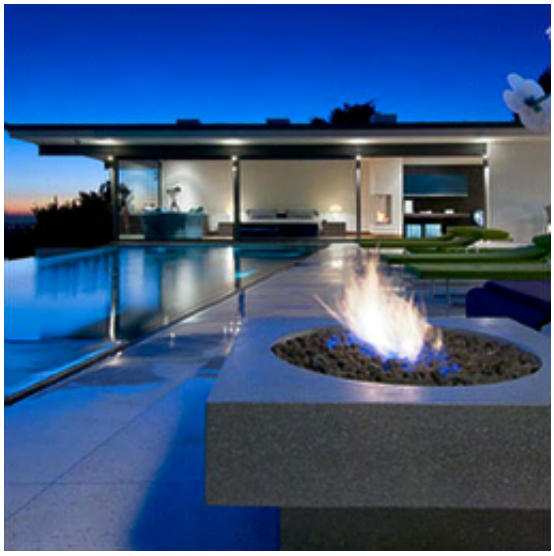
(c) $len = 5,\ dir = 120$



(d) $len = 10,\ dir = 360$

Figure 30: This figure shows 4 sample images obtained by applying the deblurring operation using Lucy-Richardson's method [34, 35] to images corrupted by motion blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

14-15. Joint deblurring and denoising using Lucy-Richardson's method [34,35]: Lucy-Richardson's method (using the *deconvlucy* function in Matlab) can also be applied to restoring images that are corrupted by additive noise, in addition to blur. We apply Lucy-Richardson's method to images corrupted by 1) Gaussian blur and additive Gaussian noise, and 2) linear motion blur and additive Gaussian noise. For Gaussian blur, we set the kernel size (s_k) to be a function of the standard deviation (σ_k) of the blur kernel: $s_k = 3 \times \sigma_k + 2$.
Parameters: Gaussian blur kernel standard deviation: $\sigma_k \in [0.5, 2]$, the motion blur kernel length: $len \in [3, 10]$, the motion blur kernel direction: $dir \in [0, 360]$, and additive Gaussian noise variance: $\sigma_g^2 \in [10^{-5}, 8 \times 10^{-4}]$)
Relevance: These distortions are representative of the artifacts caused by a number of image restoration algorithms.



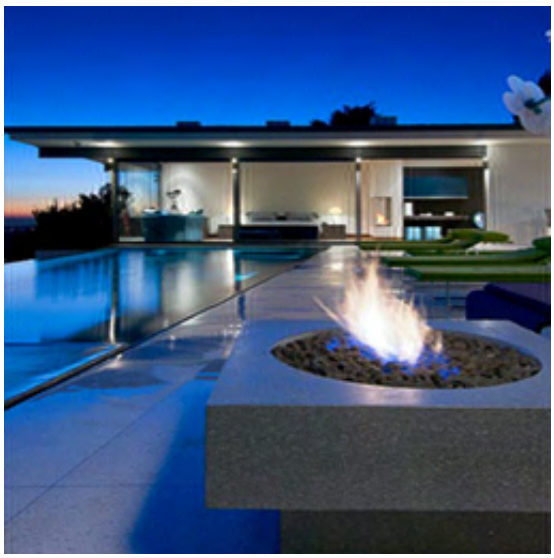
(a) $\sigma_k = 0.5,\ \sigma_g^2 = 10^{-5}$

(b) $\sigma_k = 1,\ \sigma_g^2 = 2 \times 10^{-4}$

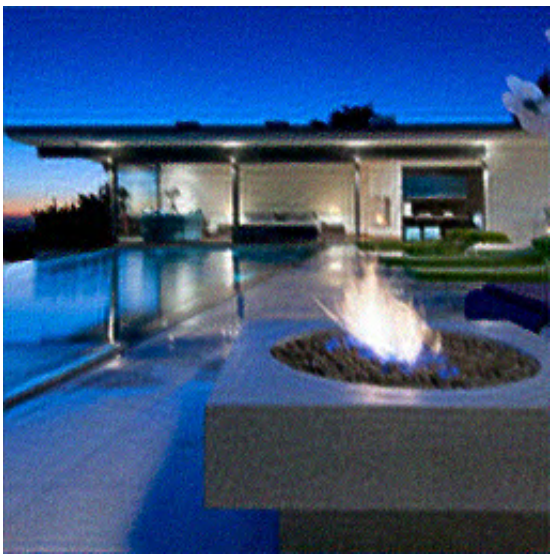
(c) $\sigma_k = 1.5,\ \sigma_g^2 = 4 \times 10^{-4}$

(d) $\sigma_k = 2,\ \sigma_g^2 = 6 \times 10^{-4}$

Figure 31: This figure shows 4 sample images obtained by applying joint deblurring and denoising using Lucy-Richardson's method [34, 35] to images corrupted by additive Gaussian noise and Gaussian blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.



(a) $len = 3,\ dir = 0,\ \sigma_g^2 = 10^{-5}$



(b) $len = 5,\ dir = 120,\ \sigma_g^2 = 2 \times 10^{-4}$

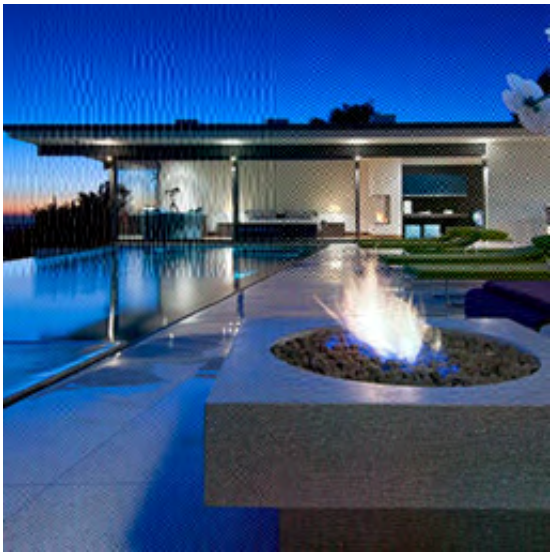
(c) $len = 7,\ dir = 240,\ \sigma_g^2 = 4 \times 10^{-4}$

(d) $len = 10,\ dir = 360,\ \sigma_g^2 = 6 \times 10^{-4}$

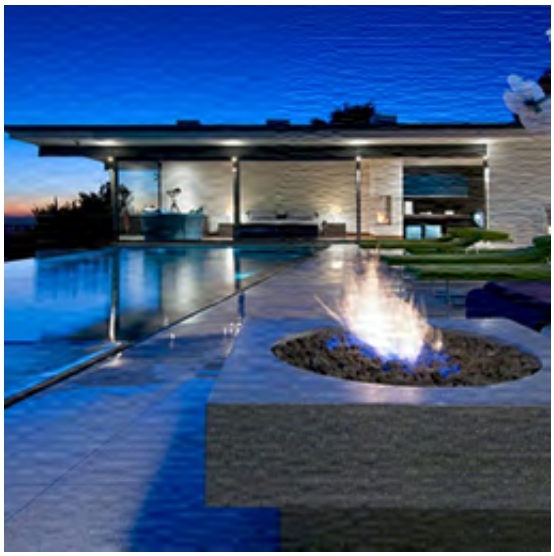
Figure 32: This figure shows 4 sample images obtained by applying joint deblurring and denoising using Lucy-Richardson's method [34,35] to images corrupted by additive Gaussian noise and linear motion blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

16. JPEG2000 transmission errors: A JPEG-2000 compressed image bitstream is QPSK-modulated to give the complex message signal $x_{\text{in}} = x_R + j \times x_I$, which is then to be transmitted. The channel model is a Rayleigh flat-fading channel with the complex-valued transfer function $h = h_R + j \times h_I$. During the transmission, the signal is corrupted by additive Gaussian noise to give the received signal y_{out} with real and imaginary parts given by $\text{Re}(y_{\text{out}}) = \text{Re}(h \times x_{\text{in}}) + n_g$ and $\text{Im}(y_{\text{out}}) = \text{Im}(h \times x_{\text{in}}) + n_g$, respectively, where $n_g \sim \mathcal{N}(0, \sigma_g^2)$ and σ_g is determined from the specified received signal-to-noise ratio (SNR). The corrupted signal y_{out} is then demodulated, resulting in the distorted image.
Parameter: Received SNR $\in [34, 42]$
Relevance: This distortion is common during the transmission of compressed images over noisy channels.

(a) Received SNR = 42



(b) Received SNR = 39



(c) Received SNR = 38

(d) Received SNR = 36

Figure 33: This figure shows 4 sample images corrupted by JPEG2000 transmission errors. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

17. Lossy compression of an image corrupted by additive Gaussian noise: We use the implementation described in [1,4], which we briefly sumamrize as follows. The input reference image is first corrupted by an additive Gaussian noise. We then use a DCT-based coder, ADCT [45], to compress this noisy image to obtain the output distorted image.
Parameters: Additive Gaussian noise standard deviation: $\sigma_g \in [0.02, 0.1]$ and the quantization step for the ADCT coder: $\Delta \in \{50, 51, ..., 160\}$
Relevance: This distortion can occur when noisy images are compressed.

(a) $\sigma_g = 0.02,\ \Delta = 50$

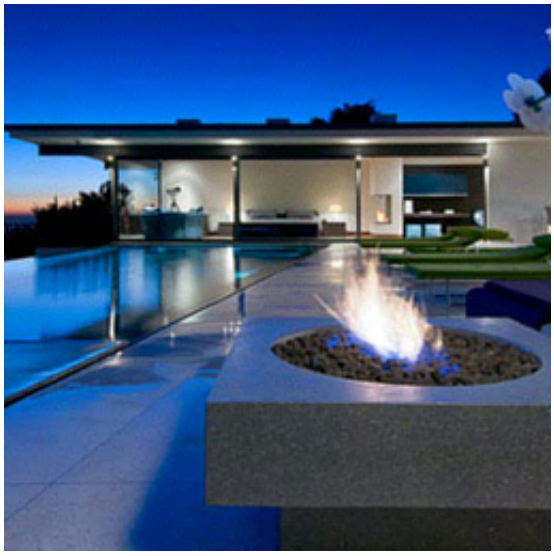
(b) $\sigma_g = 0.06,\ \Delta = 120$

(c) $\sigma_g = 0.04,\ \Delta = 80$

(d) $\sigma_g = 0.1,\ \Delta = 160$

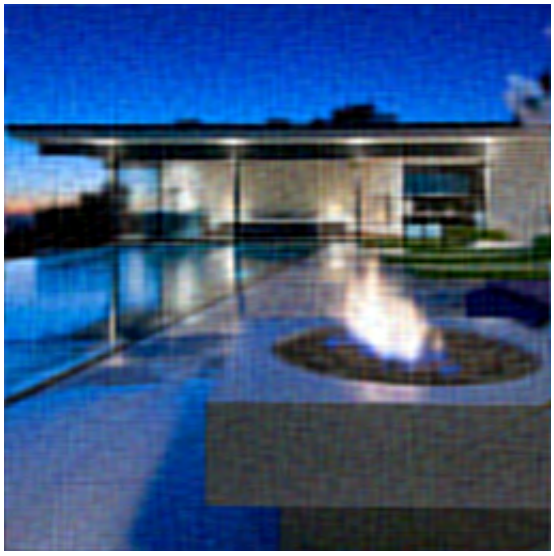
Figure 34: This figure shows 4 sample images obtained by applying lossy compression to images corrupted by additive Gaussian noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

18. Zeroing out frequency components from an image: The distorted image is generated by zeroing out the frequency components whose magnitude response fall below a threshold. The threshold is specified in terms of a percentile value (e.g., all frequency components with magnitudes smaller than the 99.4^{th} percentile are removed in Fig. 35d).
Parameter: The percentile value for computing the magnitude threshold: $per \in [60, 99.4]$
Relevance: This distortion can be caused by image compression and image filtering.



(a) $per = 60$

(b) $per = 85$



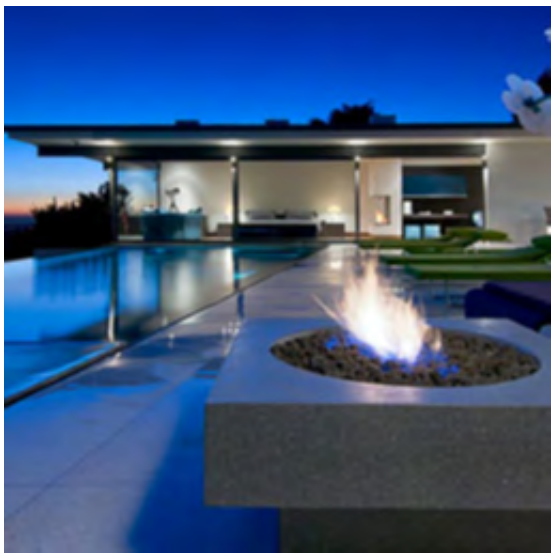
(c) $per = 94$

(d) $per = 99.4$

Figure 35: This figure shows 4 sample images corrupted by the distortion involving zeroing out frequency components from an image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

6.1.4 Detail loss

19. Gaussian blur: The distorted image is generated by convolving the undistorted reference image with a Gaussian blur kernel. We set the kernel size (s_k) to be a function of the standard deviation (σ_k) of the blur kernel: $s_k = 3 \times \sigma_k + 2$.
Parameters: Standard deviation of the blur kernel: $\sigma_k \in [0.5, 3.1]$
Relevance: This distortion can be caused by image filtering and multi-scale image processing.



(a) $\sigma_k = 0.5$

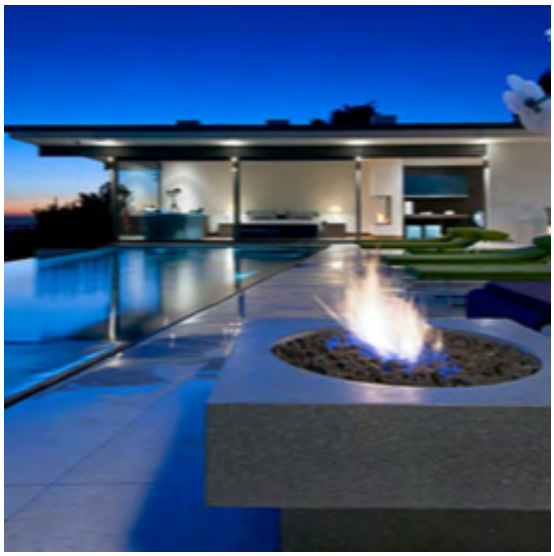
(b) $\sigma_k = 1.3$

(c) $\sigma_k = 2.1$

(d) $\sigma_k = 3.1$

Figure 36: This figure shows 4 sample images corrupted by Gaussian blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

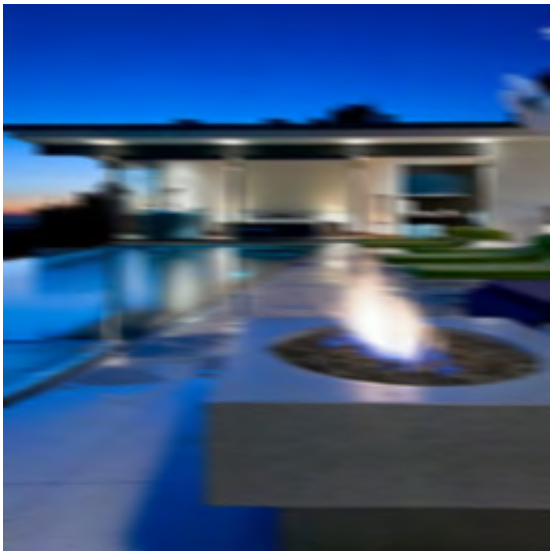
20. Linear motion blur: Linear motion blur is applied to the reference image using Matlab's *fspecial* function.
Parameters: The motion blur kernel length: $len \in [2, 10]$ and the motion blur kernel direction: $dir \in [0, 360]$
Relevance: This distortion can occur during image acquisition when the camera is in motion.



(a) $len = 2,\ dir = 0$

(b) $len = 7,\ dir = 235$

(c) $len = 4,\ dir = 115$

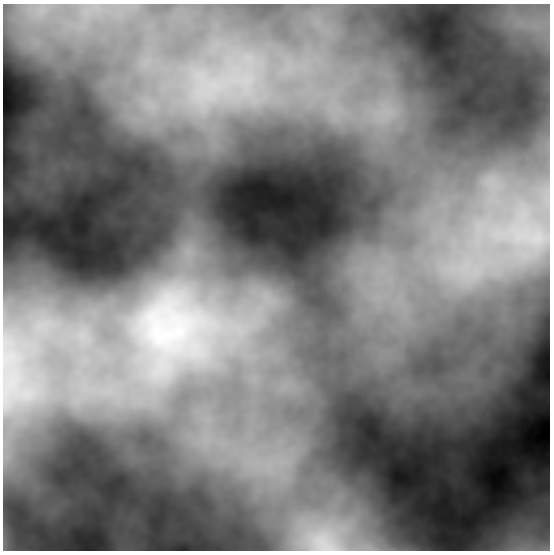


(d) $len = 10,\ dir = 355$

Figure 37: This figure shows 4 sample images corrupted by linear motion blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

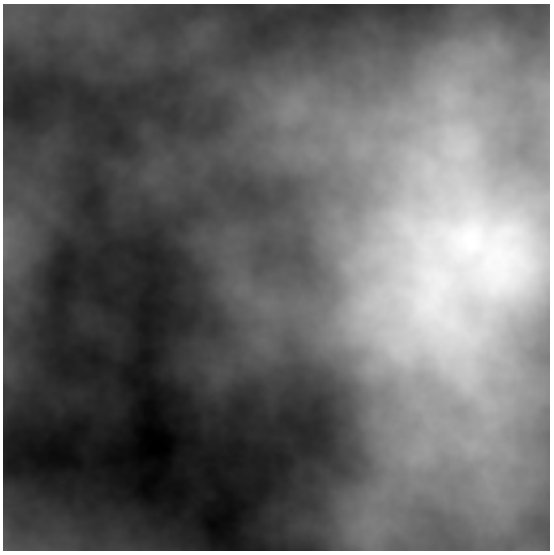
21. Locally-varying blur: The reference image is corrupted by a blur kernel that has a spatially-varying standard deviation. The standard deviation map for achieving this per-pixel blur effect is computed in the form of a Perlin noise pattern [46], which is of the same size as the reference image. The range of values of the Perlin noise pattern are linearly scaled from 0 to a pre-specified maximum.
Parameters: The maximum scale for Perlin noise generator: $s_{\max} \in [5, 8]$ and the maximum value of Perlin noise map: $v_{\max} \in [1.5, 5]$
Relevance: This distortion can occur in depth-of-field blur simulations.

(a) $v_{\max} = 3.5,\ s_{\max} = 7$



(b) Perlin noise pattern used for image in 38a.

(c) $v_{\max} = 5,\ s_{\max} = 8$

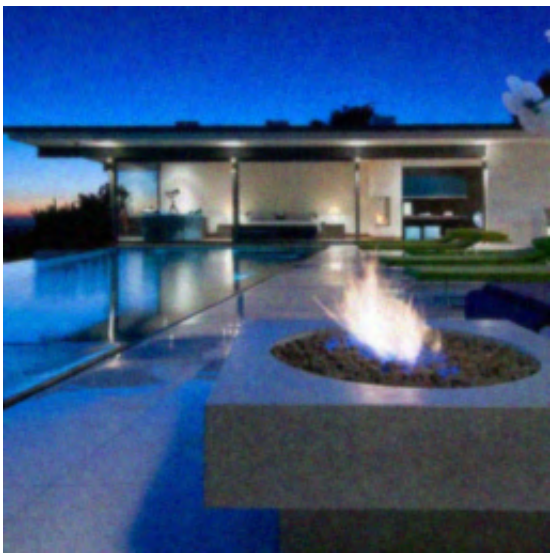


(d) Perlin noise pattern used for image in 38c.

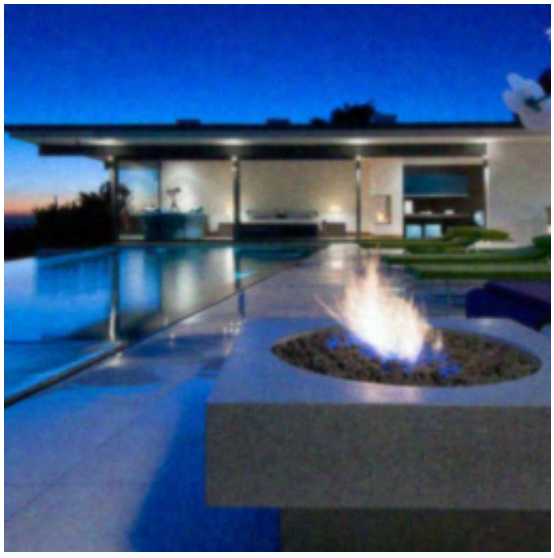
Figure 38: This figure shows 2 sample images corrupted by locally-varying blur, along with the corresponding underlying Perlin noise patterns which are used to compute the locally-varying standard deviation for the Gaussian blur kernel. The parameter settings used to generate each of the Perlin noise patterns are stated in the caption of each sub-figure.

22. Perona-Malik denoising [36]: The reference image is first corrupted by an additive Gaussian noise. We then apply Perona-Malik denoising algorithm to the noisy image. The denoised image is the output distorted image.
Parameter: Additive Gaussian noise variance: $\sigma_g^2 \in [10^{-4}, 3.1 \times 10^{-3}]$
Relevance: This distortion is representative of the artifacts caused by an important class of denoising algorithms that use anisotropic diffusion.

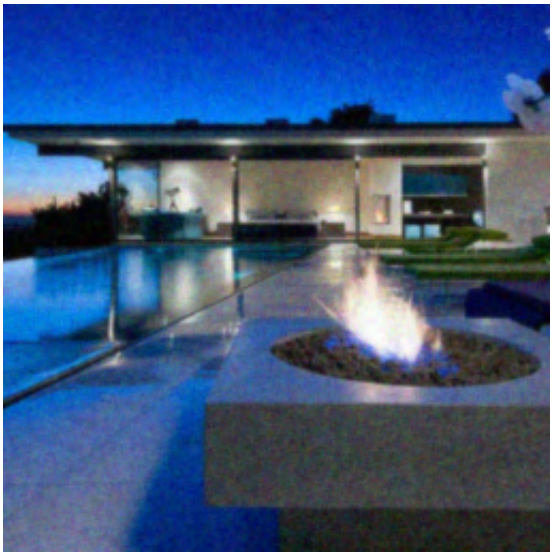
(a) $\sigma_g^2 = 10^{-4}$



(b) $\sigma_g^2 = 2.1 \times 10^{-3}$



(c) $\sigma_g^2 = 1.1 \times 10^{-3}$



(d) $\sigma_g^2 = 3.1 \times 10^{-3}$

Figure 39: This figure shows 4 sample images obtained by applying Perona-Malik denoising [36] to images corrupted by additive Gaussian noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

23. Rudin-Osher-Fatemi (ROF) denoising [37]: The reference image is first corrupted by an additive Gaussian noise. We then apply a total variation-based denoising algorithm (ROF denoising using a fixed-point iteration solver) to the noisy image. The denoised image is the resultant distorted image.
Parameter: Additive Gaussian noise variance: $\sigma_g^2 \in [0, 4 \times 10^{-3}]$
Relevance: This distortion is representative of the artifacts caused by an important class of total variation-based denoising algorithms.

(a) $\sigma_g^2 = 0.002$

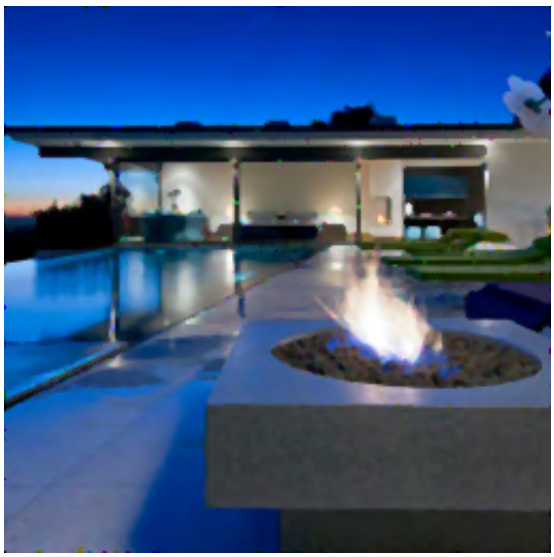
(b) $\sigma_g^2 = 0.0028$

(c) $\sigma_g^2 = 0.0036$

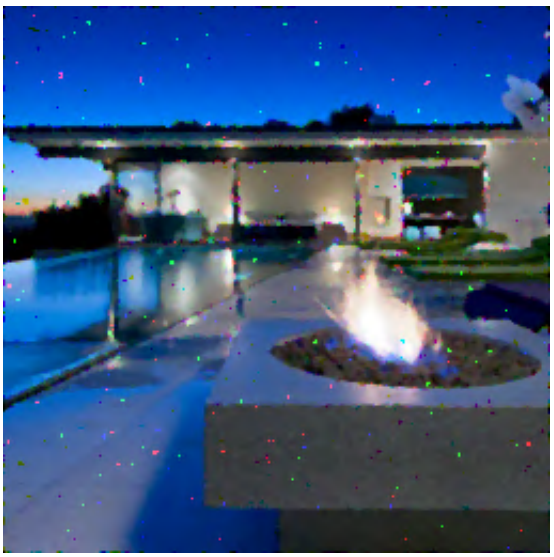
(d) $\sigma_g^2 = 0.004$

Figure 40: This figure shows 4 sample images obtained by applying ROF denoising [37] to images corrupted by additive Gaussian noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

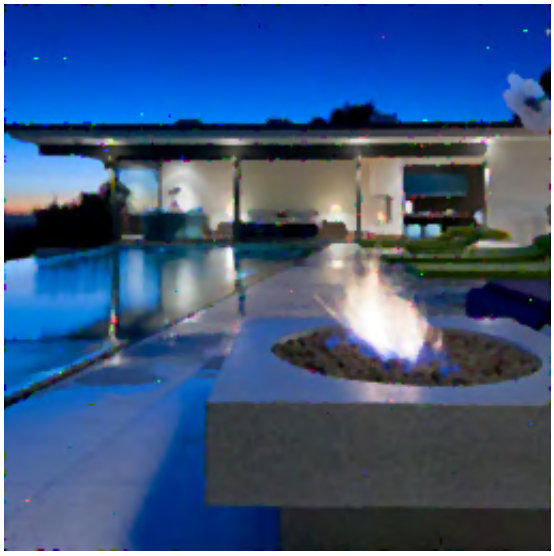
24. Median denoising of images corrupted by salt and pepper noise: The undistorted reference image is first corrupted by salt and pepper noise. We then apply a median filter to the noisy image. The filter kernel is set to a fixed size of 3×3.
Parameter: Density of salt and pepper noise: $d \in [0.08, 0.3]$
Relevance: This is a common procedure of removing salt and pepper noise.



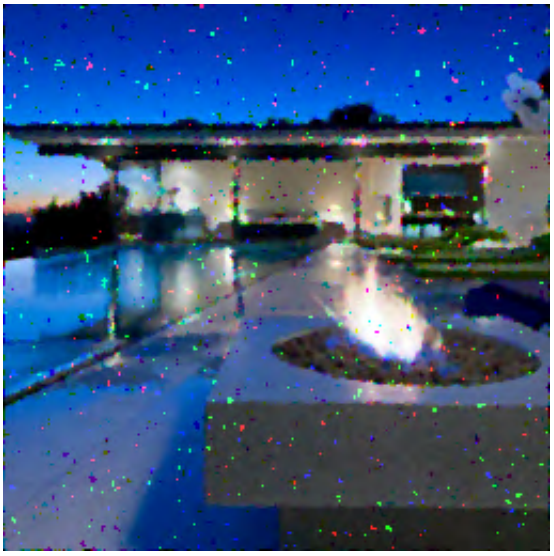
(a) $d = 0.08$



(b) $d = 0.22$



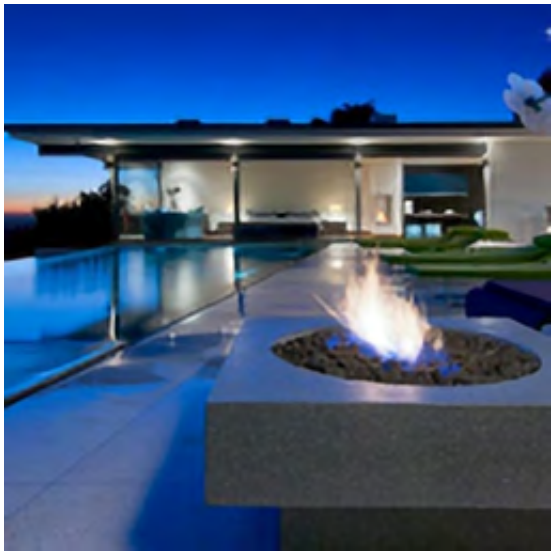
(c) $d = 0.14$



(d) $d = 0.3$

Figure 41: This figure shows 4 sample images obtained by applying median denoising to images corrupted by salt and pepper noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

25. Deep network-based super-resolution with a sparse prior [38]: An image is first downsampled by a resizing factor and then upscaled to the original size using Wang et al.'s method [38]. We use this method as one instance from the large set of super-resolution algorithms. The upscaling factor can be as high as 8 in order to bring out the artifacts caused specifically by super-resolution methods. (A small upscaling factor will make the distortion look like a common Gaussian blur.)
Parameter: The upscaling factor: $u \in \{2, 3, ..., 8\}$
Relevance: This distortion is one instance from the large set of super-resolution algorithms.



(a) $u = 2$

(b) $u = 3$

(c) $u = 6$

(d) $u = 8$

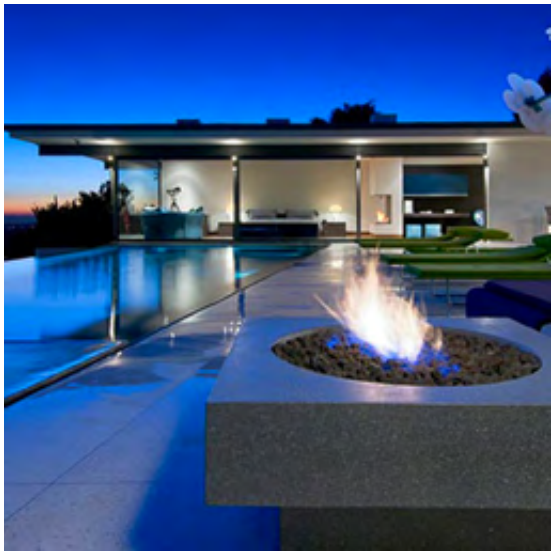
Figure 42: This figure shows 4 sample images obtained by applying super-resolution to downsampled versions of the reference image using Wang et al.'s method [38]. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

6.1.5 Color change

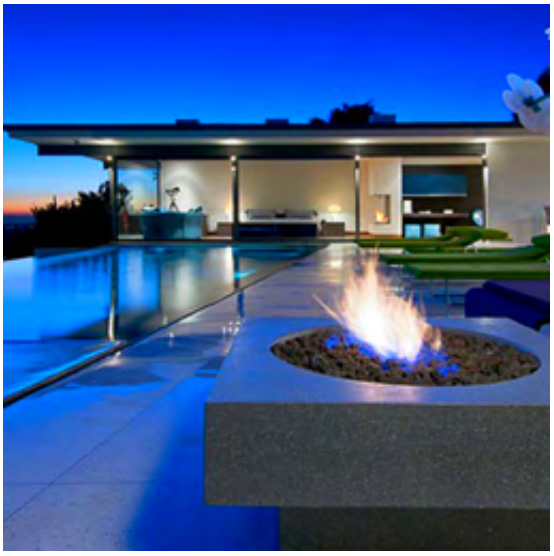
26. Change of color saturation: This distortion is implemented in the same way as in TID 2013 [4]: the RGB image is transformed into YCbCr space and the chroma components are transformed as follows: $\mathrm{Cb} = 128 + (\mathrm{Cb} - 128) \times K$ and $\mathrm{Cr} = 128 + (\mathrm{Cr} - 128) \times K$. Larger values of K result in more extreme color saturation effects.
Parameter: $K \in [0.01, 1.8]$
Relevance: This distortion can be caused by color correction algorithms and the image acquisition process.

(a) $K = 0.21$

(b) $K = 0.51$



(c) $K = 1.2$



(d) $K = 1.6$

Figure 43: This figure shows 4 sample images obtained by changing the color saturation of the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

27. Contrast stretching: Contrast changing is performed as follows: $I_D(x,y,c) = \frac{1}{1+(\frac{\bar{I}_c}{I_R(x,y,c)+\epsilon})^{\alpha}}$, where I_D is the distorted image, I_R is the reference image, and $\bar{I}_c$ is the mean intensity for channel c. As α increases, the distorted image shows more contrast (with a larger range of pixel values), while as α decreases, the range of pixel values is compressed. Fig. 44 shows the contrast stretch transformation with different values of α.
Parameter: $\alpha \in [0.5, 3.8]$
Relevance: Contrast stretching is a commonly-used technique for image color enhancement.

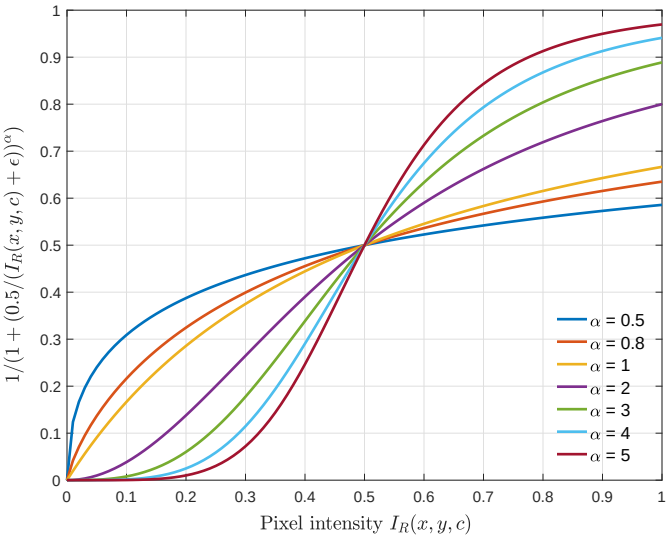
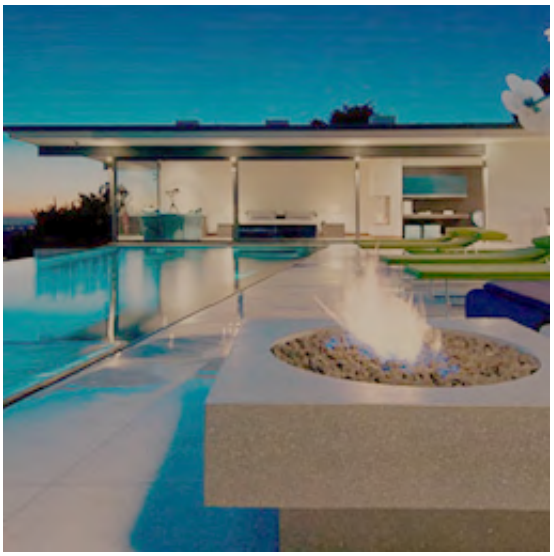
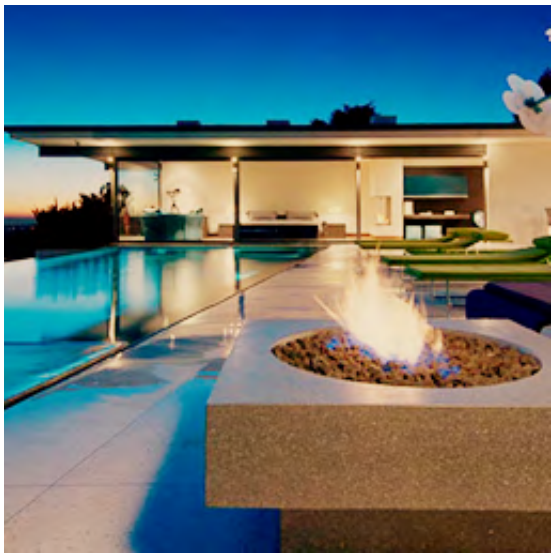


Figure 44: Contrast stretch transformation curve with different values of α, assuming (without loss of generality) that $\bar{I}_c = 0.5$.

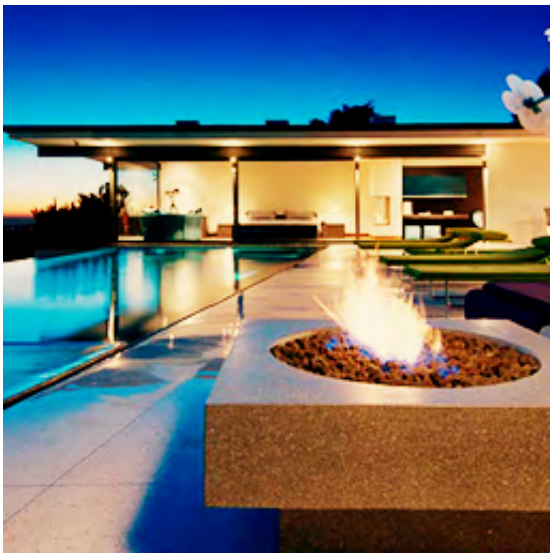
(a) $\alpha = 0.5$



(b) $\alpha = 0.8$



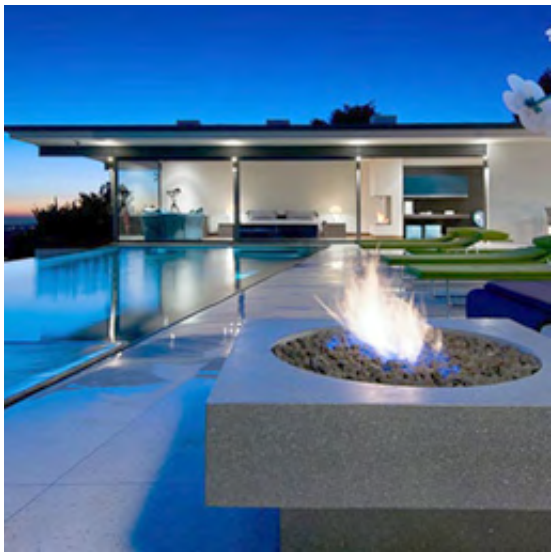
(c) $\alpha = 1.8$



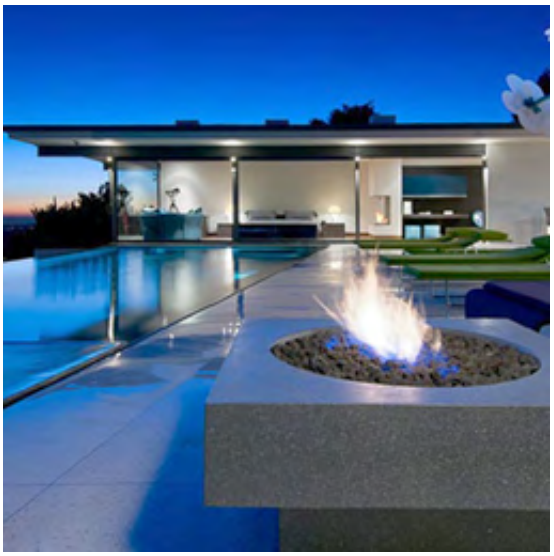
(d) $\alpha = 2.8$

Figure 45: This figure shows 2 sample images obtained by applying the contrast stretch transformation to the reference image. The parameter settings used to generate each of the 2 images are stated in the caption of each sub-figure.

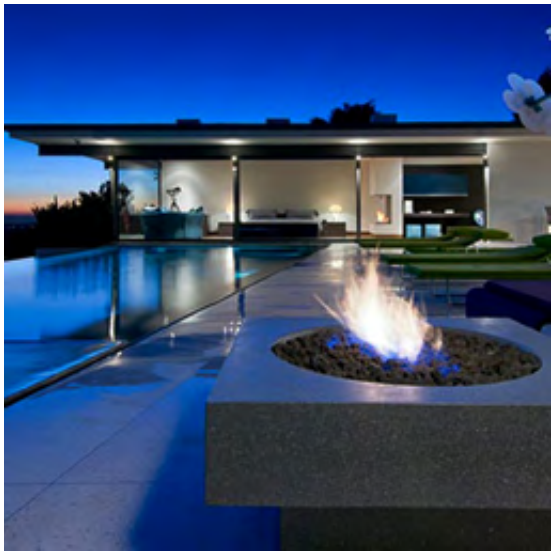
28. Gamma transformation: The distorted image is obtained by using the following formula: $I_D(x, y, c) = I_R(x, y, c)^\gamma$, where I_R is the reference image.
Parameter: $\gamma \in [0.5, 1.7]$
Relevance: This is a commonly-used technique for image color correction.



(a) $\gamma = 0.6$



(b) $\gamma = 0.7$

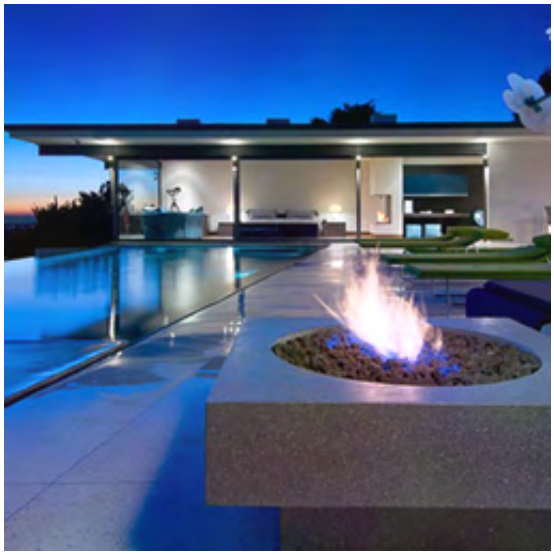


(c) $\gamma = 1.05$

(d) $\gamma = 1.25$

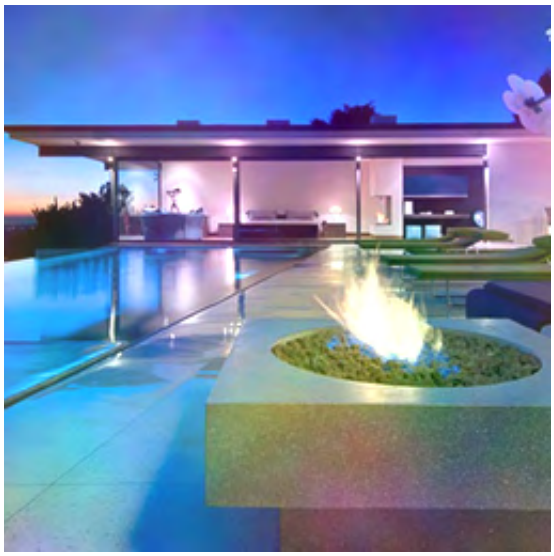
Figure 46: This figure shows 4 sample images obtained by applying gamma transformation to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

29. Local color shift: Pixel values at each pixel in reference image (I_R) are shifted to give the locally-varying color effect. Instead of independently shifting the pixel value at each location of I_R (which would look like additive noise), the color shift map is computed in the form of a Perlin noise pattern [46] independently for each channel (maximum scale of the pattern is fixed to 8). Each location of the Perlin noise pattern indicates the amount of the shift at that pixel location. The range of values of the Perlin noise pattern are linearly scaled from 0 to a pre-specified maximum. The resultant pixel values are clamped to $[0, 1]$.
Parameters: The maximum value (corresponding to the maximum color shift) to be obtained from Perlin noise pattern: $v_{\max} \in [0.1, 0.5]$
Relevance: This distortion can be caused by artistic image processing, flaws in camera image processing pipelines, and some color correction algorithms (e.g., local white balancing, dehazing).

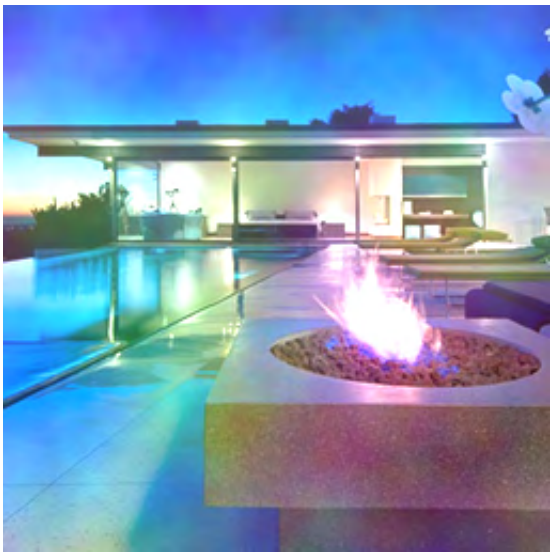


(a) $v_{\max} = 0.1$

(b) $v_{\max} = 0.3$



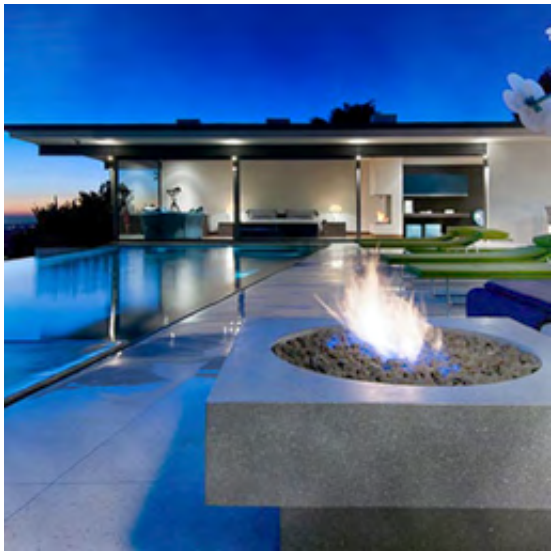
(c) $v_{\max} = 0.4$



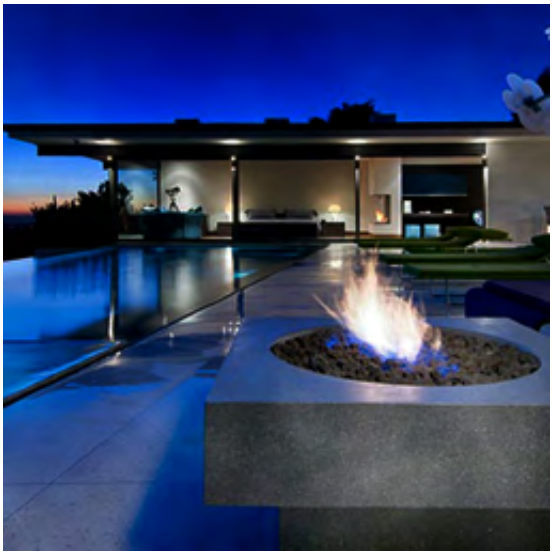
(d) $v_{\max} = 0.5$

Figure 47: This figure shows 4 sample images obtained by applying local color shifts to the reference image. The per-channel Perlin noise pattern used to guide the amount of per-pixel shift is evident from the color patterns on each image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

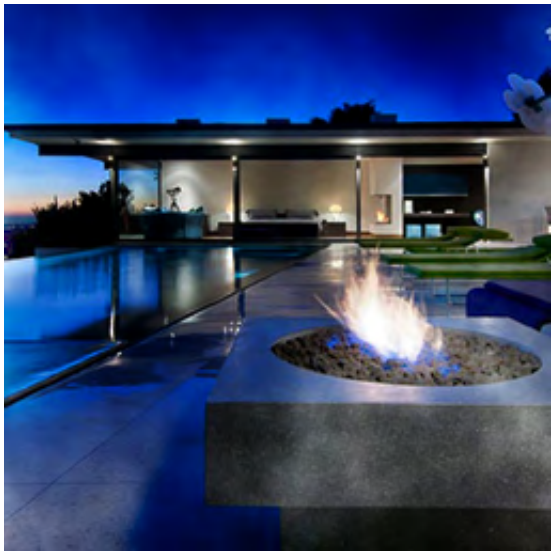
30. Local contrast change: A distorted image is generated by having a unique α for contrast stretching (see description of contrast stretching (training distortion No. 27) for the transformation formula) for each pixel (x, y, c). A Perlin noise pattern is used to specify the α at each pixel (maximum scale of the pattern is fixed to 8). The minimum and maximum value of Perlin noise pattern are pre-specified.
Parameters: The minimum value of Perlin noise pattern: $v_{\min} \in [0.2, 0.7]$ and the maximum value of Perlin noise pattern: $v_{\max} \in [1.2, 3]$
Relevance: The distorted image has spatially varying contrast, which is a common artifact of color correction algorithms (e.g., dehazing).



(a) $v_{\min} = 0.3,\ v_{\max} = 1.2$



(b) $v_{\min} = 0.5,\ v_{\max} = 2.2$

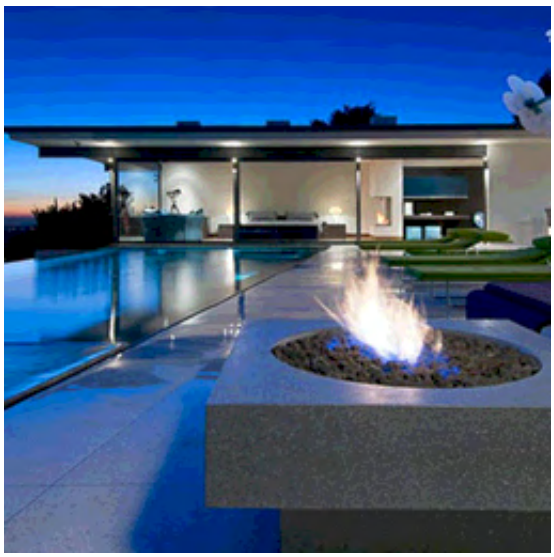


(c) $v_{\min} = 0.2,\ v_{\max} = 2.7$

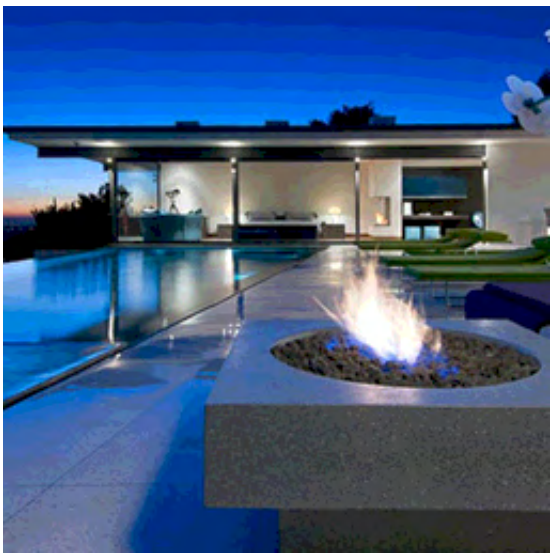
(d) $v_{\min} = 0.3,\ v_{\max} = 2.7$

Figure 48: This figure shows 4 sample images obtained by applying local contrast changes to the reference image. The Perlin noise pattern used to guide compute per-pixel α is evident from the color patterns on each image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

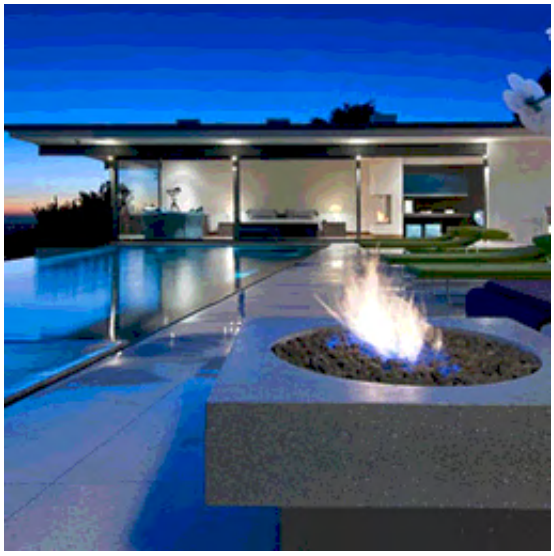
31. Color quantization: A distorted image is generated by first computing the intensity segmentation thresholds using Otsu's segmentation method (Matlab's *multithresh* method) and then quantizing the image based on these thresholds (Matlab's *imquantize* function).
Parameter: The number of intensity levels to be used for quantization: $L \in [4, 20]$
Relevance: Quantization is a crucial step in several applications including compression and segmentation.



(a) $L = 18$



(b) $L = 16$



(c) $L = 12$

(d) $L = 6$

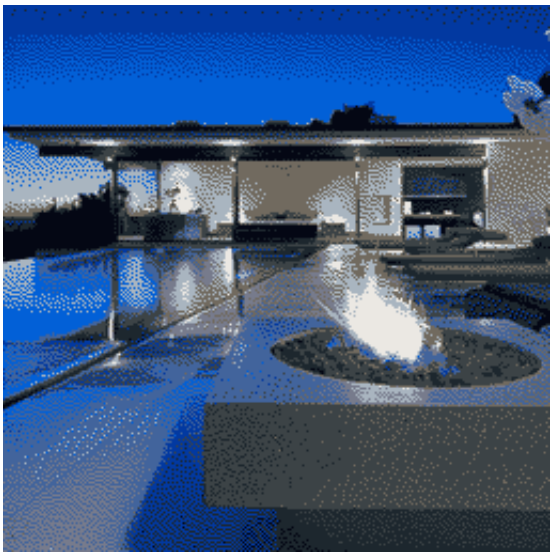
Figure 49: This figure shows 4 sample images corrupted by image quantization. The parameter setting used to generate each of the 4 images are stated in the caption of each sub-figure.

32. Color quantization with dither: Matlab function *rgb2ind* is used to implement this distortion, which converts an RGB image into a quantized image with dithering.
Parameter: The number of intensity levels to be used for quantization: $L \in [10, 95]$
Relevance: This distortion is typical in image printing [4].

(a) $L = 80$

(b) $L = 60$

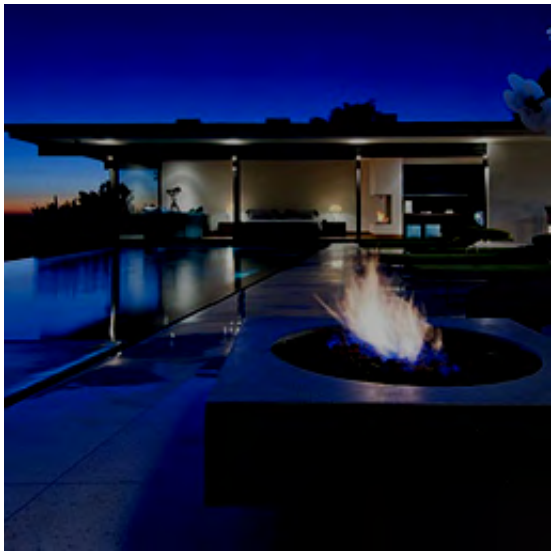
(c) $L = 20$



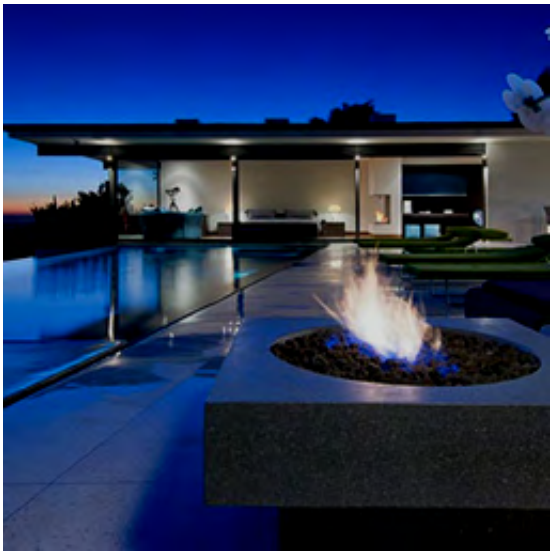
(d) $L = 10$

Figure 50: This figure shows 4 sample images corrupted by image quantization with dithering. The parameter setting used to generate each of the 4 images are stated in the caption of each sub-figure.

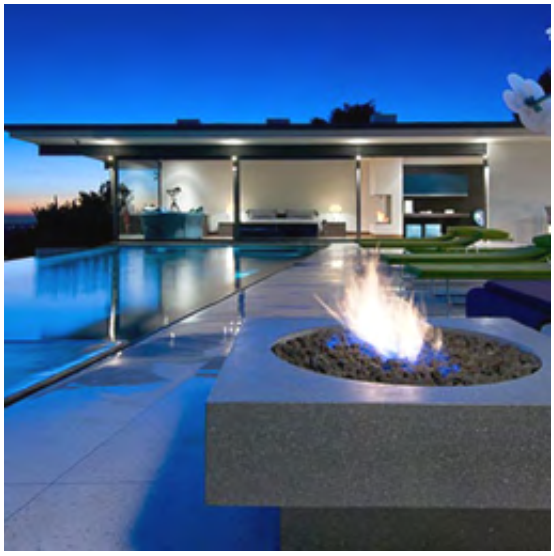
33. Mean color shift: To obtain a distorted image, the color values in each channel of the reference image are shifted by a constant (same shift value for all channels). The pixel values of the resultant image are then clamped to $[0, 1]$.
Parameter: The intensity shift value: $v_s \in [-0.3, 0.3]$
Relevance: This distortion can be caused during the image acquisition process or during color correction of images.



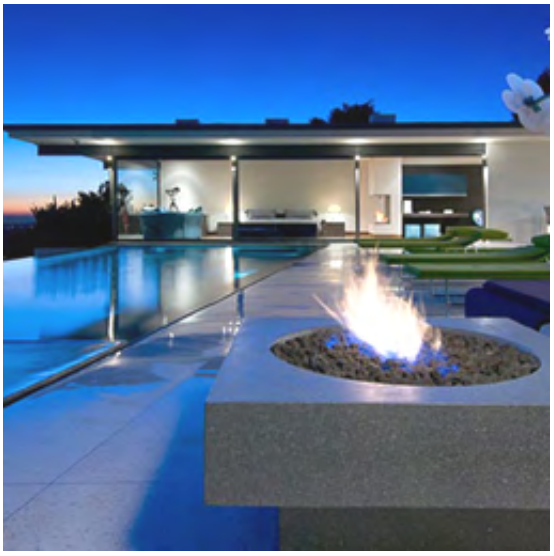
(a) $v_s = -0.3$



(b) $v_s = -0.15$



(c) $v_s = 0.05$



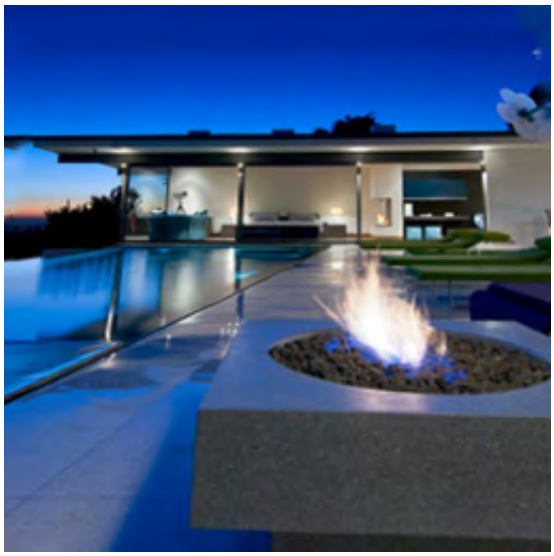
(d) $v_s = 0.1$

Figure 51: This figure shows 4 sample images corrupted by mean color shift. The parameter setting used to generate each of the 4 images are stated in the caption of each sub-figure.

6.1.6 Geometric transformations

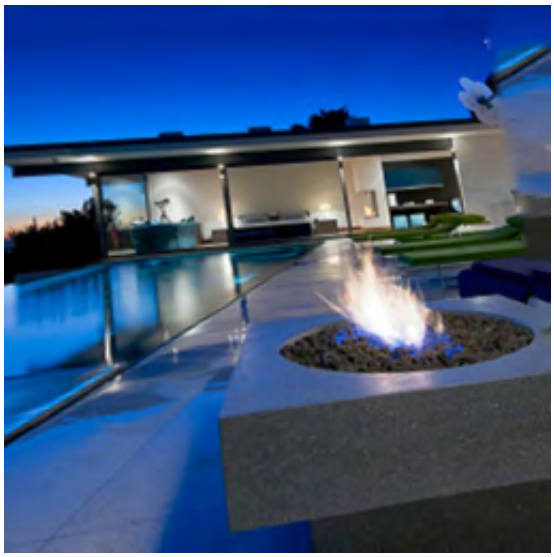
NOTE: To maintain the size of the resulting distorted image from each of the following geometric transformations and to simulate the real-world cases in which such distortions would be expected to occur, the geometrically distorted images are hole-filled in a content-aware manner near the image borders. The hole-filling is done using an open-source tool *GMIC* [47]. In most of the cases, our geometrically distorted images do not reveal too much empty space around the image borders and as a result, the hole-filled images simulate the continuous world that is captured by a camera.

34. Projective transformation: For this distortion, we simulate a projective transformation by transforming the image coordinates with a homography matrix. To construct this matrix, we begin with certain simplifying assumptions about the camera matrix for reference image, $\mathbf{C_0}$. A camera matrix is given by $\mathbf{C} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]$. Here, $\mathbf{K}$ is the intrinsic matrix and $[\mathbf{R} \mid \mathbf{t}]$ is the extrinsic matrix comprising of rotation ($\mathbf{R}$, which can be computed as a product of rotation matrices along the x, y, z-axes: $\mathbf{R} = \mathbf{R_x}\mathbf{R_y}\mathbf{R_z}$) and translation ($\mathbf{t}$) information about the camera within the world coordinates. Without loss of generality, we assume the intrinsic matrix $\mathbf{K_0}$ for $\mathbf{C_0}$ to be a 3×3 identity matrix, and extrinsic parameters to have no rotation or translation, which gives $\mathbf{C_0} = [\mathbf{I}_{3\times3} \mid \mathbf{0}]_{3\times4}$. We then compute a new camera matrix, $\mathbf{C_1} = \mathbf{K_1}[\mathbf{R_1} \mid \mathbf{t_1}]$ by modifying the rotation matrix (extrinsic parameters, modified by changing rotation angles about x, y, z axes: $\theta_x, \theta_y, \theta_z$ in degrees respectively) and adding a skew factor (intrinsic parameter, s_q). The resultant 2D projective transformation is then given by: $\mathbf{H} = \mathbf{K_1}\mathbf{R_1}$ which is used to transform the reference image [48].
Parameters: $\theta_x \in [-0.05, 0.05]$, $\theta_y \in [-0.05, 0.05]$, $\theta_z \in [-8, 8]$, and $s_q \in [-0.18, 0.18]$
Relevance: A common procedure in 3D image processing and computer vision applications (e.g., panoramic stitching, 3D reconstruction).



(a) $\theta_x = 0.05,\ \theta_y = 0.01,\ \theta_z = -2,\ s_q = -0.03$

(b) $\theta_x = 0.05,\ \theta_y = -0.01,\ \theta_z = -5,\ s_q = -0.03$

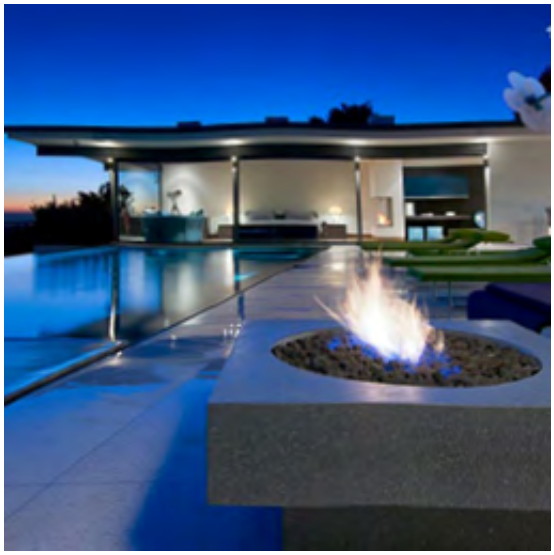


(c) $\theta_x = 0.03,\ \theta_y = -0.04,\ \theta_z = 7,\ s_q = 0.02$

(d) $\theta_x = -0.04,\ \theta_y = -0.02,\ \theta_z = -8,\ s_q = 0.07$

Figure 52: This figure shows 4 sample images obtained by applying projective transformation to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

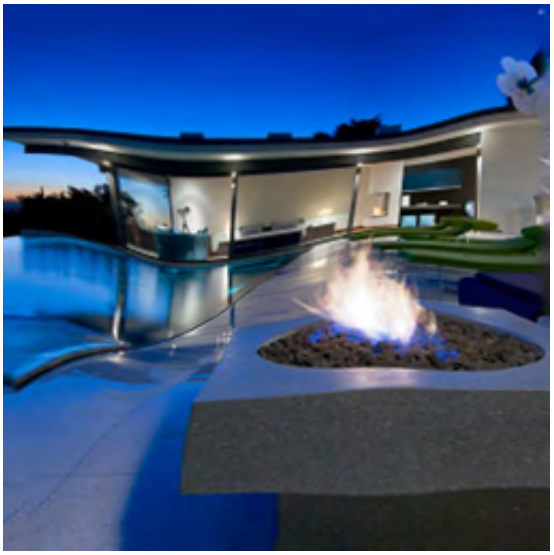
35. Warping (using a local spatial shift map): Locally-varying spatial shift is applied to each pixel in the reference image. The per-pixel shift map is computed at a coarse scale (s_w), and then upsampled (using bicubic interpolation). At the coarse scale, the spatial shift values are drawn from a zero-mean uniform distribution with a specified width w. The upsampled shift map is used to warp the reference image.
Parameters: $s_w \in [1, 20]$ and $w \in [0.05, 0.5]$
Relevance: This distortion can occur during correspondence map-based image transformations (e.g., generalized PatchMatch [49]).



(a) $s_w = 3,\ w = 0.3$

(b) $s_w = 5,\ w = 0.125$

(c) $s_w = 10,\ w = 0.15$

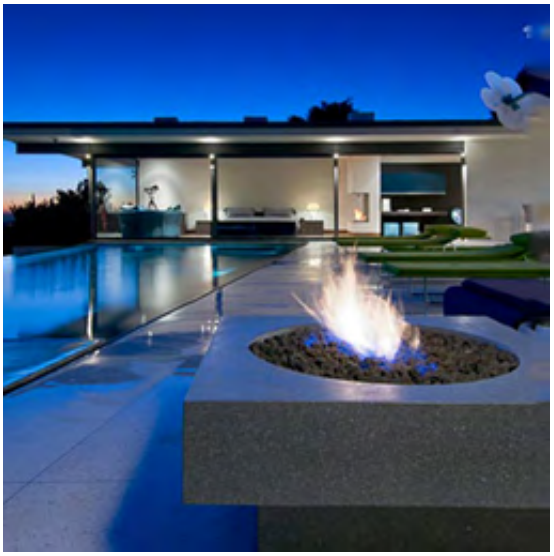


(d) $s_w = 20,\ w = 0.35$

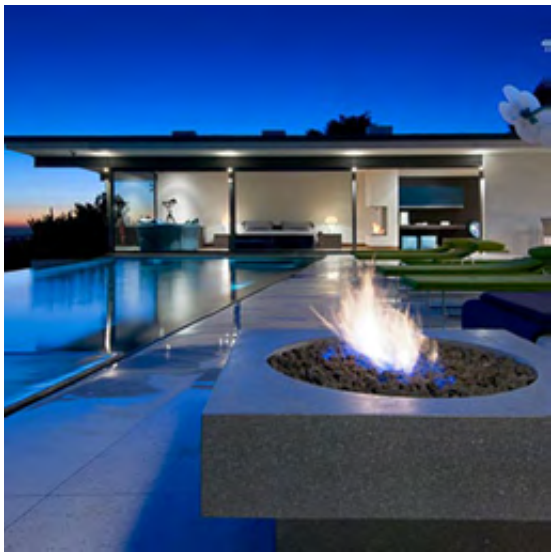
Figure 53: This figure shows 4 sample images obtained by applying warping (using a local spatial shift map) to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

36. Spatial shift: The reference image is shifted horizontally and/or vertically.
Parameters: horizontal shift: $s_h \in [-10, 10]$ and vertical shift: $s_v \in [-10, 10]$
Relevance: A common procedure in 3D image processing and computer vision applications (e.g., panoramic stitching, 3D reconstruction).

(a) $s_h = -9,\ s_v = -7$



(b) $s_h = -10,\ s_v = -1$

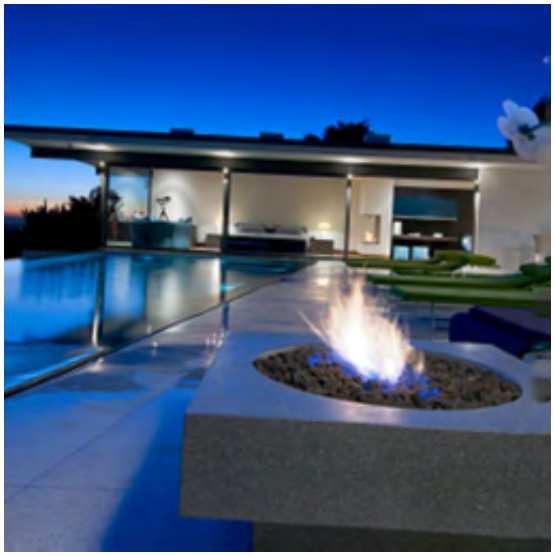


(c) $s_h = -2,\ s_v = 5$

(d) $s_h = 5,\ s_v = 1$

Figure 54: This figure shows 4 sample images obtained by applying global spatial shifts to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

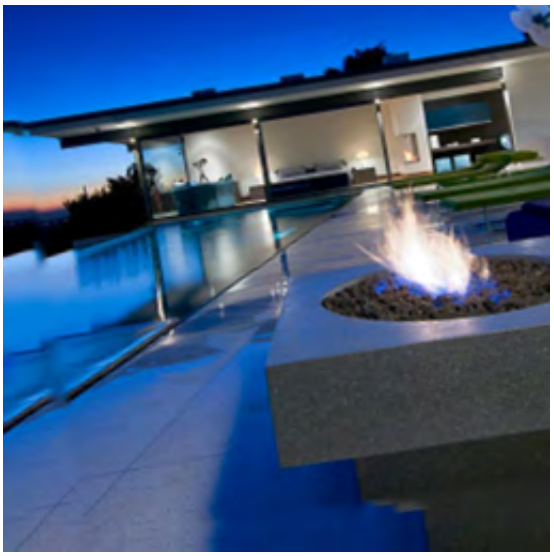
37. 2D rotation: The reference image is rotated using a 2D rotation matrix.
Parameter: Angle of rotation: $\theta \in [-9, 9]$
Relevance: A common procedure in 3D image processing and computer vision applications (e.g., panoramic stitching, 3D reconstruction).



(a) $\theta = 3$

(b) $\theta = 5$

(c) $\theta = 8$



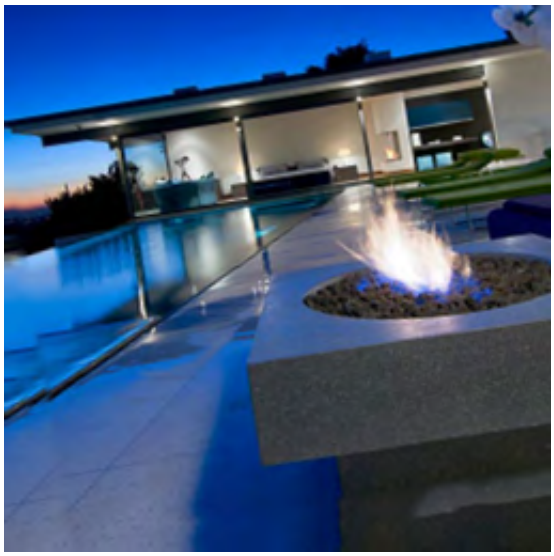
(d) $\theta = -9$

Figure 55: This figure shows 4 sample images obtained by applying global image rotation to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

38. Spatial shift and rotation: A combination of rotation and shift is applied to the reference image.
Parameters: horizontal shift: $s_h \in [-10, 10]$, vertical shift: $s_v \in [-10, 10]$, and the angle of rotation: $\theta \in [-9, 9]$
Relevance: A common procedure in 3D image processing and computer vision applications (e.g., panoramic stitching, 3D reconstruction).

(a) $s_h = 6,\ s_w = -6,\ \theta = 9$

(b) $s_h = 10,\ s_w = -1,\ \theta = 8$



(c) $s_h = -9,\ s_w = -1,\ \theta = -9$

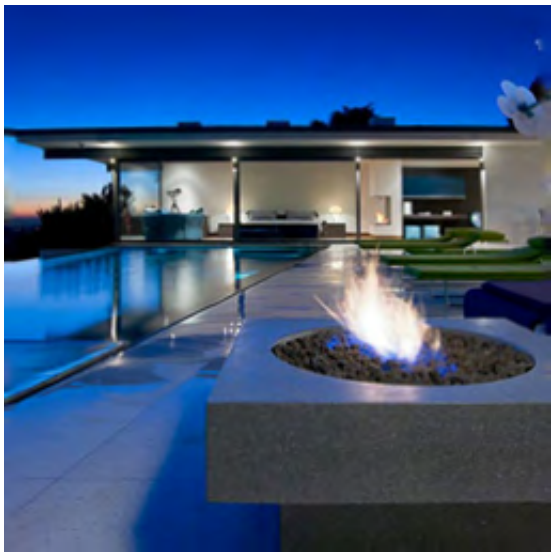
(d) $s_h = 10,\ s_w = 8,\ \theta = -5$

Figure 56: This figure shows 4 sample images obtained by applying global image shift and rotation to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

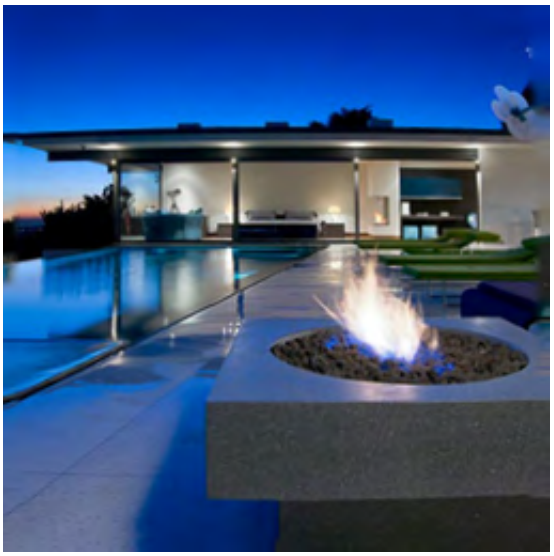
39. Radial barrel transformation: Barrel distortion (in polar coordinates) is formulated as:

$$r' = r \times (1 + a \times r^2) \text{ and } \theta' = \theta, \tag{9}$$

where (r, θ) are the original polar coordinates, (r', θ') are the transformed polar coordinates. For barrel transformation, $a > 0$.
Parameter: $a \in [10^{-6}, 5.5 \times 10^{-6}]$
Relevance: This distortion simulates a common lens distortion artifact during the image acquisition process.



(a) $a = 1.5 \times 10^{-6}$



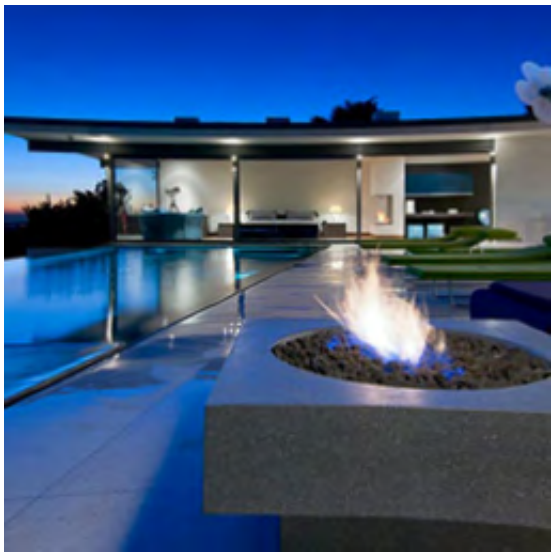
(b) $a = 3 \times 10^{-6}$

(c) $a = 3.5 \times 10^{-6}$

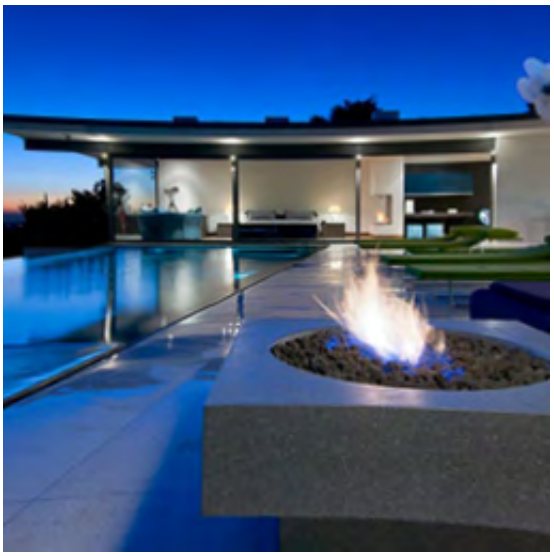
(d) $a = 5.5 \times 10^{-6}$

Figure 57: This figure shows 4 sample images obtained by applying radial barrel transformation to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

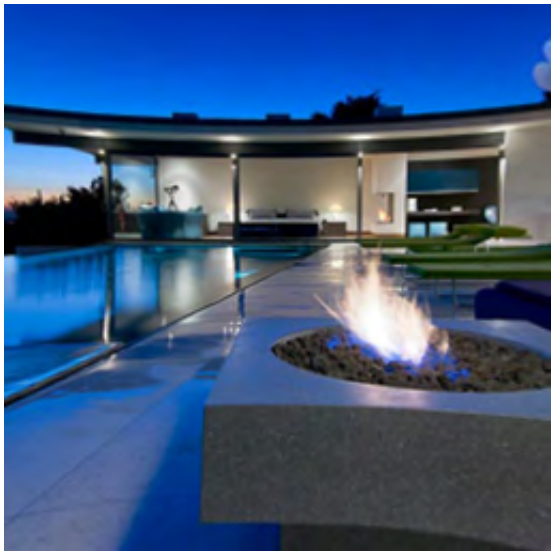
40. Radial pincushion transformation: Pincushion transformation is obtained by modifying the coefficient a in Eq. 9 (training distortion No. 39) such that $a < 0$ (and $|a| < 1$).
Parameter: $a \in [-7 \times 10^{-6}, -1 \times 10^{-6}]$
Relevance: This distortion simulates a common lens distortion artifact during the image acquisition process.



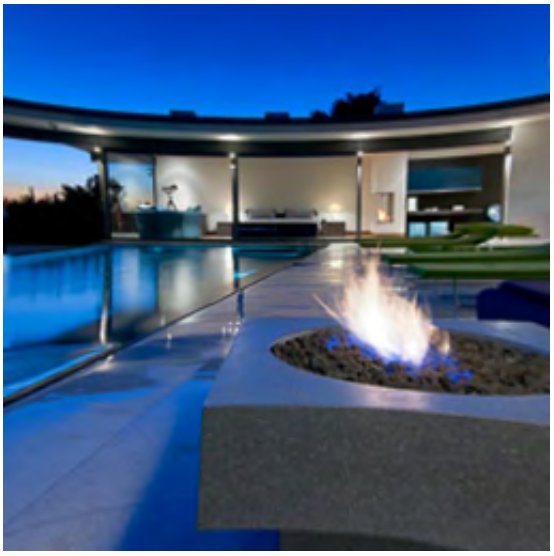
(a) $a = -2.5 \times 10^{-6}$



(b) $a = -3 \times 10^{-6}$



(c) $a = -5 \times 10^{-6}$

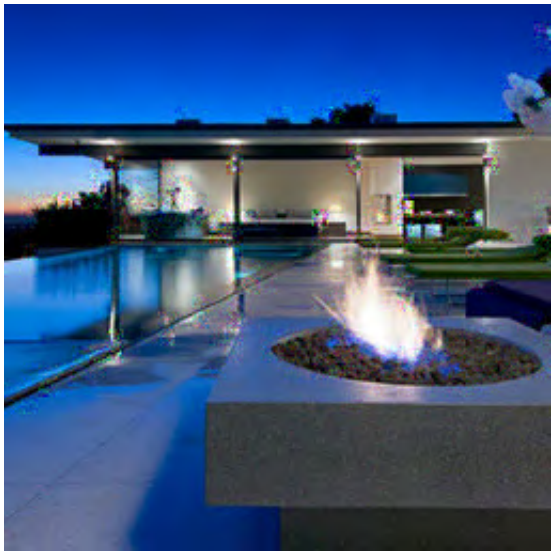


(d) $a = -6.5 \times 10^{-6}$

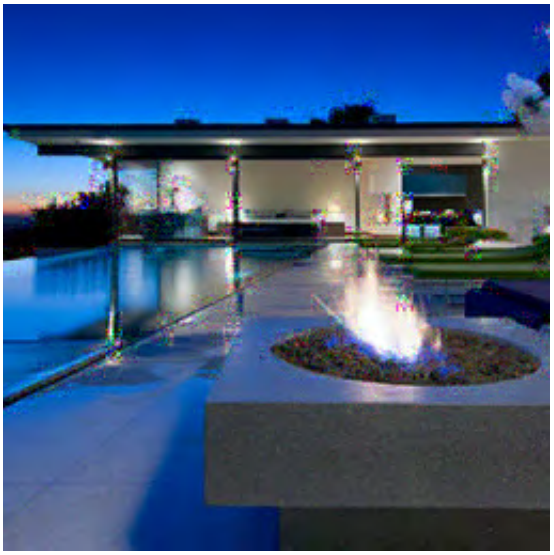
Figure 58: This figure shows 4 sample images obtained by applying radial pincushion transformation to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

6.1.7 Others

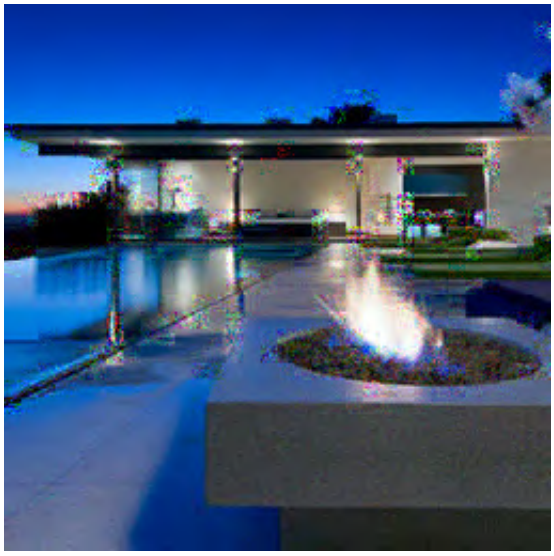
41. Compressive sensing using Orthogonal Matching Pursuit (OMP) [39]: An image is considered to be sparse in DCT basis. We use random Gaussian sampling matrix to sample the image and to reconstruct the image based on the samples. We then solve a set of linear equations under sparsity constraints by using the OMP algorithm.
Parameter: The percentage of samples used to reconstruct an image: $p_s \in [40, 80]$
Relevance: Compressive sensing, sparse reconstruction of an image



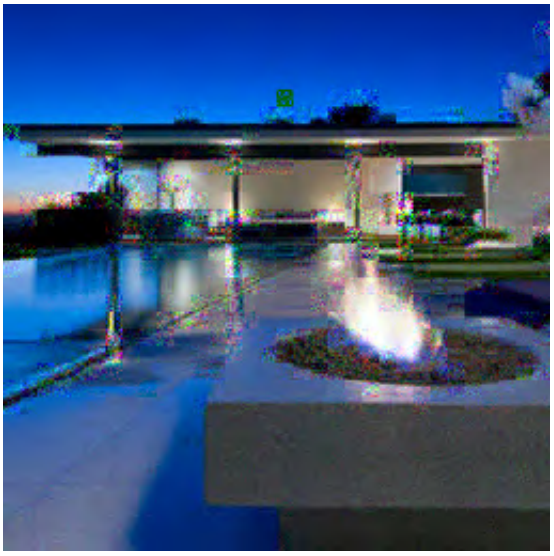
(a) $p_s = 70$



(b) $p_s = 60$



(c) $p_s = 50$



(d) $p_s = 40$

Figure 59: This figure shows 4 sample images obtained by sparse sampling and reconstructing the reference image using the OMP algorithm [39]. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

42. Chromatic aberrations: This distortion is implemented by applying independent spatial shifts, s_c, to each channel of the reference image, followed by applying Gaussian blur to each channel. We set the Gaussian blur standard deviation (σ_k) to be proportional to shift applied to a channel: $\sigma_k = 0.35 \times s_c$.
Parameters: $s_c \in [-20, 20]$
Relevance: This distortion simulates an image acquisition defect where there is a lens failure in focusing all the colors to the same point.

(a) $s_c = 2$

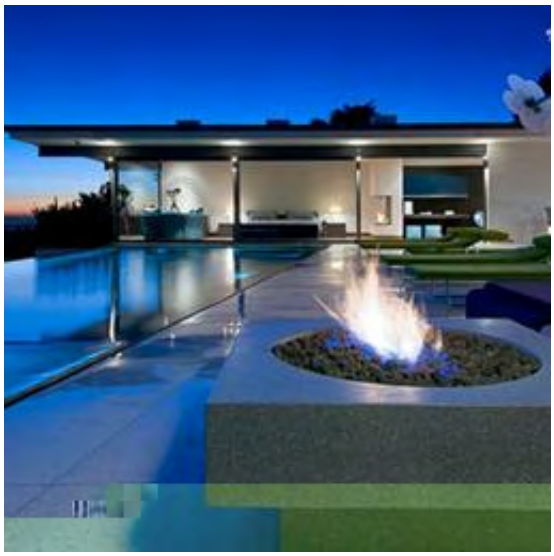
(b) $s_c = 2.75$

(c) $s_c = 3.5$

(d) $s_c = 6$

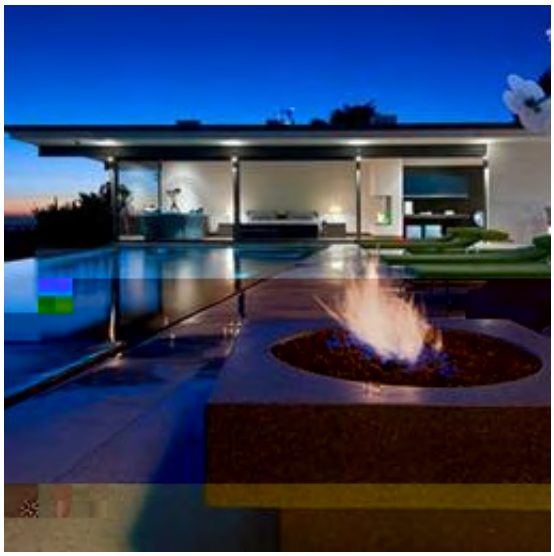
Figure 60: This figure shows 4 sample images corrupted by chromatic aberrations. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

43. JPEG transmission error: This is implemented similar to the JPEG-2000 transmission error (Sec. 6.2.2 of this file, training distortion No. 12). A JPEG-compressed image bitstream is QPSK-modulated to give the complex message signal $x_{\text{in}} = x_R + j \times x_I$, which is then to be transmitted. The channel model is a Rayleigh flat-fading channel with the complex-valued transfer function $h = h_R + j \times h_I$. During the transmission, the signal is corrupted by additive Gaussian noise to give the received signal y_{out} with real and imaginary parts given by $\text{Re}(y_{\text{out}}) = \text{Re}(h \times x_{\text{in}}) + n_g$ and $\text{Im}(y_{\text{out}}) = \text{Im}(h \times x_{\text{in}}) + n_g$, respectively, where $n_g \sim \mathcal{N}(0, \sigma_g^2)$ and σ_g is determined from the specified received signal-to-noise ratio (SNR). The corrupted signal y_{out} is then demodulated, resulting in the distorted image.
Parameter: Received SNR $\in [36, 40]$
Relevance: This distortion is common during the transmission of compressed images over noisy channels.

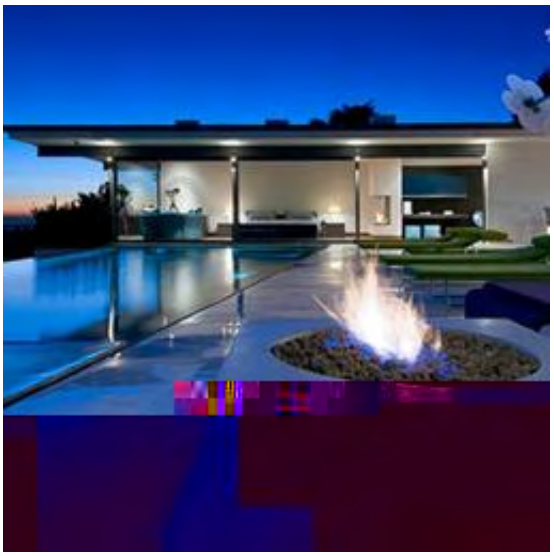


(a) Received SNR = 25

(b) Received SNR = 23



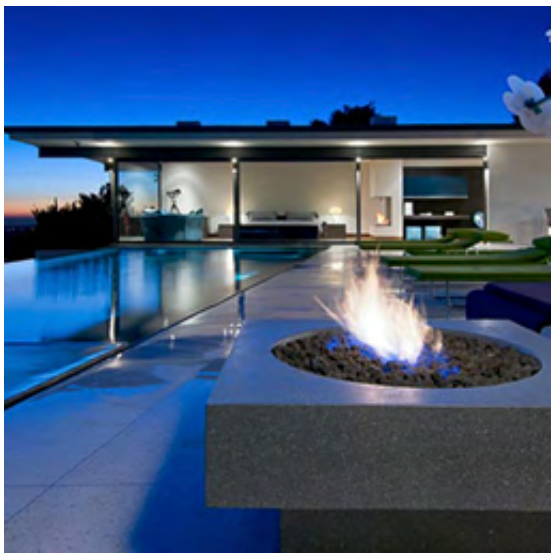
(c) Received SNR = 19



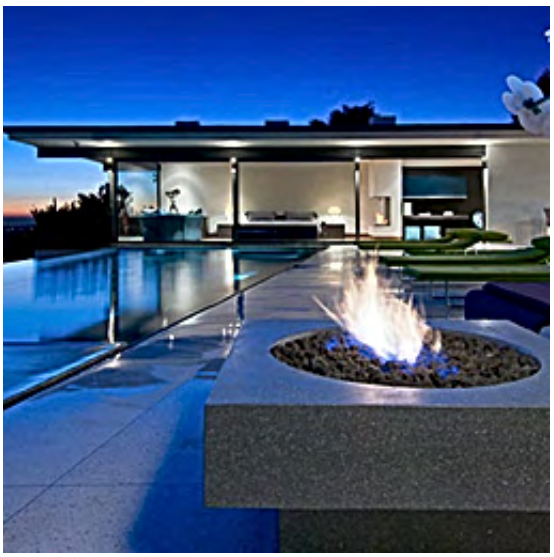
(d) Received SNR = 18

Figure 61: This figure shows 4 sample images corrupted by JPEG transmission errors. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

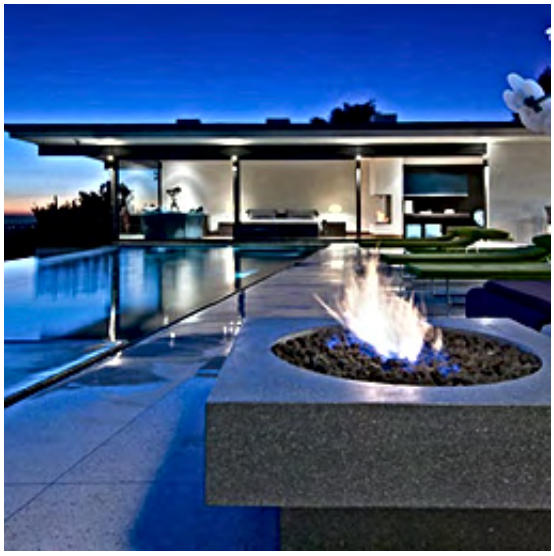
44. Image sharpening: This image sharpening operation is implemented using the *imsharpen* function in Matlab.
Parameters: The standard deviation of the Gaussian lowpass kernel: $\sigma_k \in [1, 4]$ and the strength of sharpening: $\alpha_s \in [0.05, 2.05]$
Relevance: This distortion can be caused by image enhancement operations.



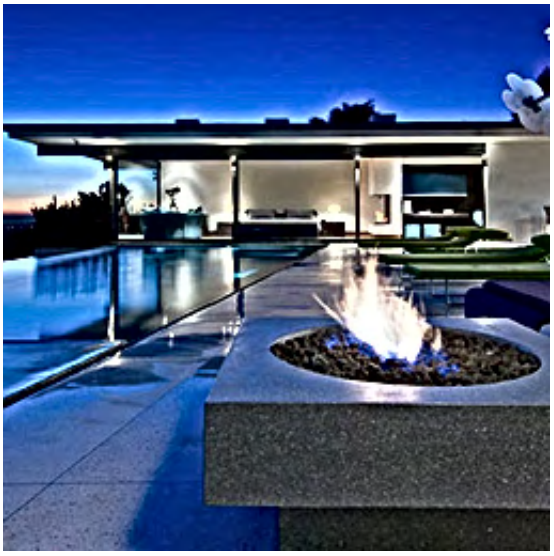
(a) $\sigma_k = 2,\ \alpha_s = 0.05$



(b) $\sigma_k = 2,\ \alpha_s = 1.05$



(c) $\sigma_k = 4,\ \alpha_s = 1.05$



(d) $\sigma_k = 4,\ \alpha_s = 2.05$

Figure 62: This figure shows 4 sample images obtained by applying image sharpening to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

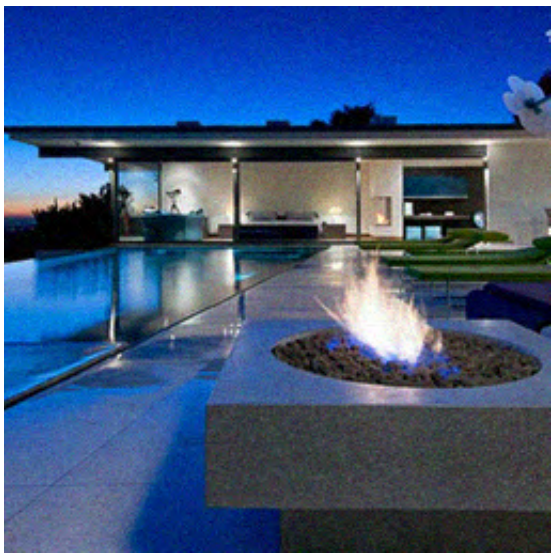
6.2 Test image distortions

Visual Effects	Unseen Test Image Distortions
Noise	1. additive Gaussian noise 2. Gaussian noise in high frequency components 3. speckle noise 4. Poisson noise
Artifacts with regular patterns	5. deblurring using Chan et al.'s method [50] for images corrupted by Gaussian blur 6. deblurring using Chan et al.'s method for images corrupted by motion blur 7. deblurring using Tikhonov regularization for images corrupted by Gaussian blur 8. deblurring using Tikhonov regularization for images corrupted by motion blur 9. joint deblurring and denoising using Chan et al.'s method for images corrupted by additive Gaussian noise and Gaussian blur 10. joint deblurring and denoising using Chan et al.'s method for image corrupted by additive Gaussian noise and motion blur 11. comfort noise 12. JPEG2000 compression
Detail loss	13. BM3D denoising [51] of images corrupted by additive Gaussian noise 14. BM3D denoising of images corrupted by spatially-varying Gaussian noise 15. ROF denoising using split Bregman solver [52] 16. compressive sensing using Danielyan et al.'s method [53] 17. super-resolution using SRCNN [54] 18. super-resolution using Peleg et al.'s method [55] 19. super-resolution using Timofte et al.'s method [56] 20. super-resolution using Zeyde et al.'s method [57] 21. soft focus
Color change	22. color temperature change 23. log transformation 24. histogram equalization 25. vignette effect
Geometric transformations	26. vertical image stretch/shrink 27. horizontal image stretch/shrink 28. swirl transformation 29. wave transformation 30. image shift and rotation and radial distortion 31. radial distortion using 2^{nd} order polynomial function

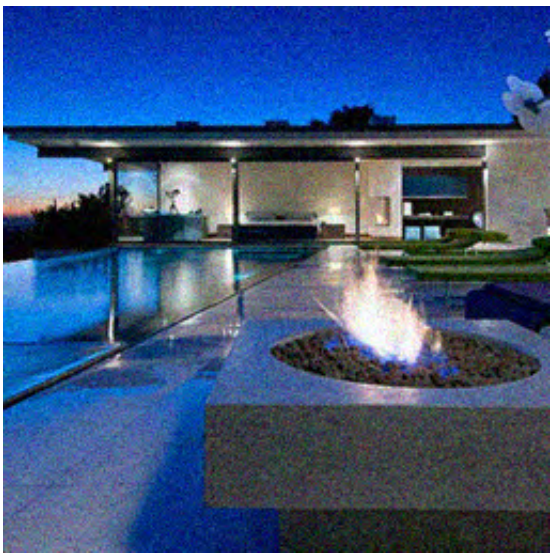
Table 4: Test distortions are categorized according to their visual effects. Implementation details and pictorial examples for each example can be found by following the hyperlink of each visual effect category.

6.2.1 Noise

1. Additive Gaussian noise: Gaussian noise of a specified variance is added to the image, resulting in a distorted image (using Matlab's *imnoise* function).
Parameter: Gaussian noise variance: $\sigma_g^2 \in [0.001, 0.024]$
Relevance: This is a common noise in image processing and may occur during the image acquisition process.



(a) $\sigma_g^2 = 0.001$



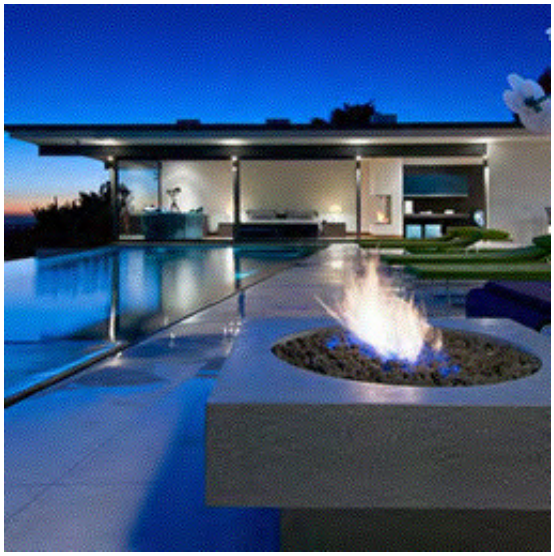
(b) $\sigma_g^2 = 0.003$

(c) $\sigma_g^2 = 0.015$

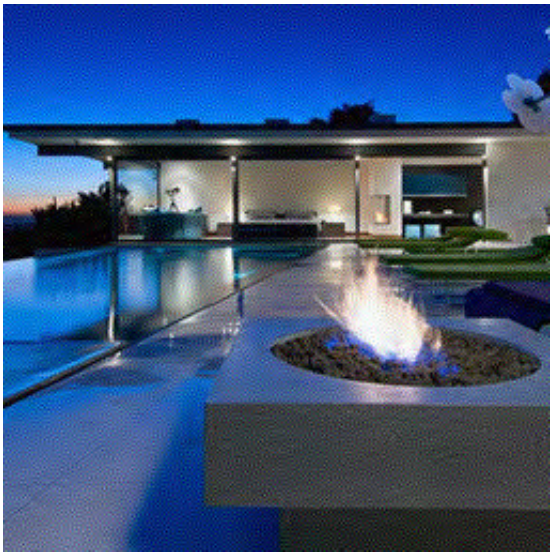
(d) $\sigma_g^2 = 0.021$

Figure 63: This figure shows 4 sample images corrupted by additive Gaussian noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

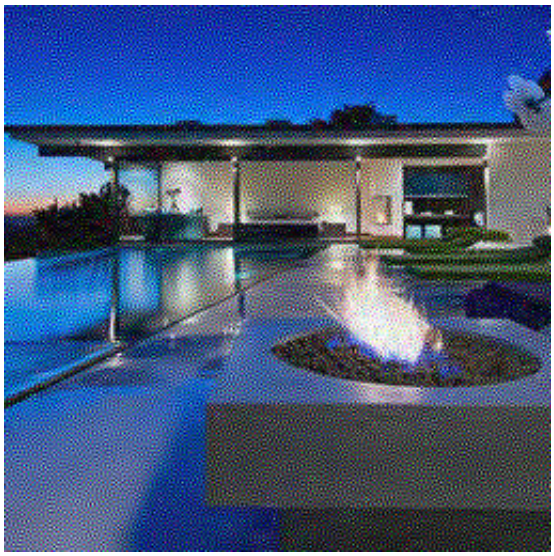
2. Gaussian noise in high frequency components: Distorted images are generated by adding Gaussian noise to high frequency components of an image. To select the high frequency components for adding noise, a Fourier transform of an image is parameterized in polar coordinates, with the image center as the origin. The components that are farther (in terms of the normalized distance from the origin) than a specified threshold are selected as high frequency components. (The points farthest from the origin have a distance of 1.)
Parameters: Gaussian noise standard deviation: $\sigma_g \in [30, 150]$ and the threshold on normalized distance for selecting frequency components to add noise to: $t \in [0.5, 0.99]$. (The pixel values range from 0 to 255 in this case.)
Relevance: This distortion is used to capture the spatial frequency sensitivity of the HVS.



(a) $\sigma_g = 30,\ t = 0.94$



(b) $\sigma_g = 40,\ t = 0.91$

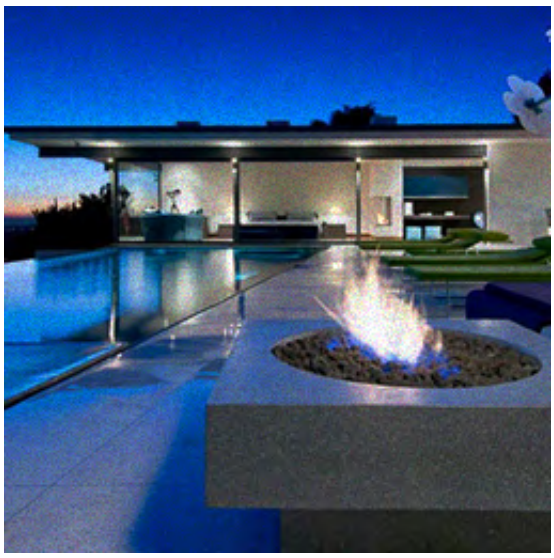


(c) $\sigma_g = 110,\ t = 0.91$

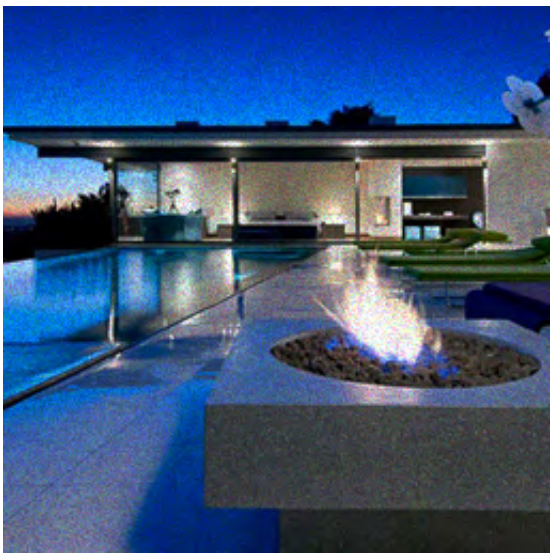
(d) $\sigma_g = 120,\ t = 0.5$

Figure 64: This figure shows 4 sample images corrupted by Gaussian noise in high frequency components. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

3. Speckle noise: An image corrupted by speckle noise is given by $I_D(x, y, c) = I_R(x, y, c) + N(x, y, c) \times I_R(x, y, c)$, where $I_D(x, y, c)$ is the distorted image, $I_R(x, y, c)$ is the reference image, and $N(x, y, c)$ is a uniformly-distributed random noise with zero mean and a specified width. This is implemented using the *imnoise* function in Matlab.
Parameter: Width of the uniform random distribution: $w \in [0.001, 0.055]$
Relevance: This distortion can occur during the acquisition of medical images or tomography images.

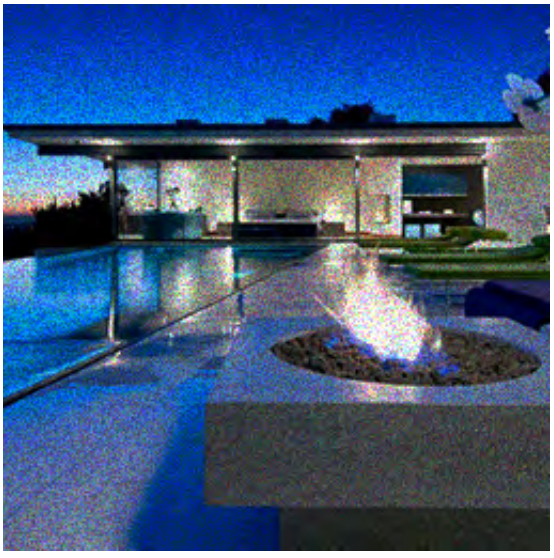


(a) $w = 0.006$



(b) $w = 0.011$

(c) $w = 0.031$



(d) $w = 0.036$

Figure 65: This figure shows 4 sample images corrupted by speckle noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

4. Poisson noise: This distortion is generated using *imnoise* function of Matlab, which generates Poisson noise based on the image pixel values. The parameter settings for this noise are pre-specified to fixed numbers in Matlab, thus we keep this distortion free of additional parameters.
Parameter: None
Relevance: This distortion occurs during the acquisition of digital images.

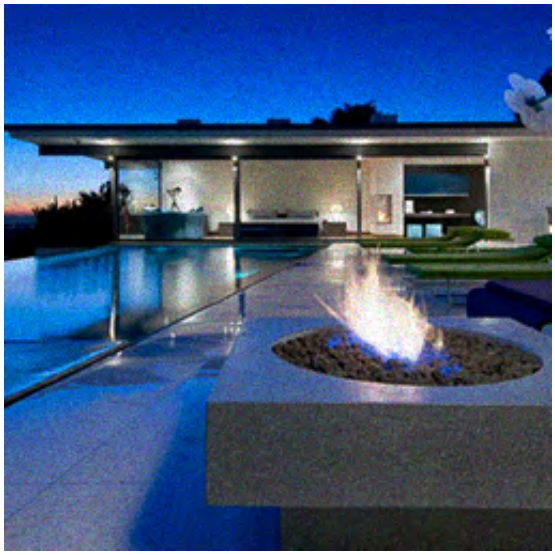


Figure 66: This figure shows a sample image corrupted by Poisson noise.

6.2.2 Artifacts with regular patterns

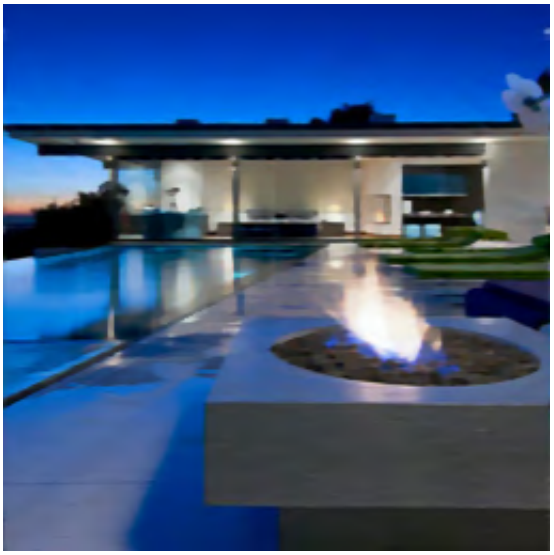
5-6. Deblurring using Chan et al.'s method [50]: We include a few more realizations of deblurring and joint deblurring and denoising algorithms in our test set, which appear visually distinct from their training set counterparts. In this distortion, we include deblurring using total variation regularization [50] for images corrupted by Gaussian or motion blur. For Gaussian blur, we set the kernel size (s_k) to be a function of the standard deviation (σ_k): $s_k = 3 \times \sigma_k + 2$.
Parameters: Gaussian blur kernel standard deviation: $\sigma_k \in [0.5, 2.5]$, motion blur kernel length: $len \in [3, 25]$, and the motion blur kernel direction: $dir \in [0, 360]$
Relevance: The distortions are representative of the artifacts caused by a number of image restoration algorithms.

(a) $\sigma_k = 0.4$ (b) $\sigma_k = 1.2$

(c) $\sigma_k = 2$ (d) $\sigma_k = 2.5$

Figure 67: This figure shows 4 sample images obtained by applying the deblurring operation using Chan et al.'s method [50] to images corrupted by Gaussian blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

(a) $len = 7,\ dir = 210$



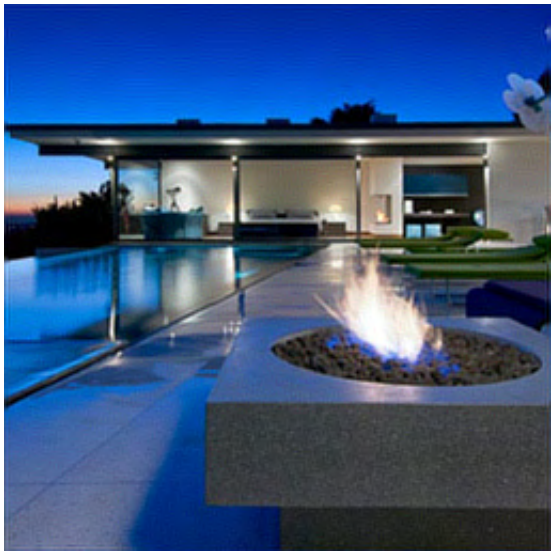
(b) $len = 16,\ dir = 180$

(c) $len = 15,\ dir = 330$

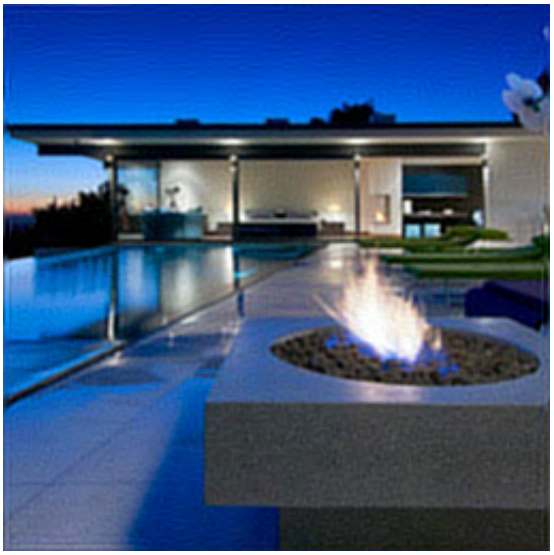
(d) $len = 22,\ dir = 300$

Figure 68: This figure shows 4 sample images obtained by applying the deblurring operation using Chan et al.'s method [50] to images corrupted by motion blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

7-8. Deblurring using Tikhonov regularization: Images corrupted by Gaussian and linear motion blur are deblurred using an algorithm with Tikhonov regularization (using the *deconvreg* function in Matlab). For Gaussian blur, we set the kernel size (s_k) to be a function of the standard deviation (σ_k): $s_k = 3 \times \sigma + 2$.
Parameters: Gaussian blur kernel standard deviation: $\sigma_k \in [0.5, 2.5]$, motion blur kernel length: $len \in [3, 10]$, and the motion blur kernel direction: $dir \in [0, 360]$.(The pixel values range from 0 to 255 in this case.)
Relevance: The distortions are representative of the artifacts caused by a number of image restoration algorithms.



(a) $\sigma_k = 1$

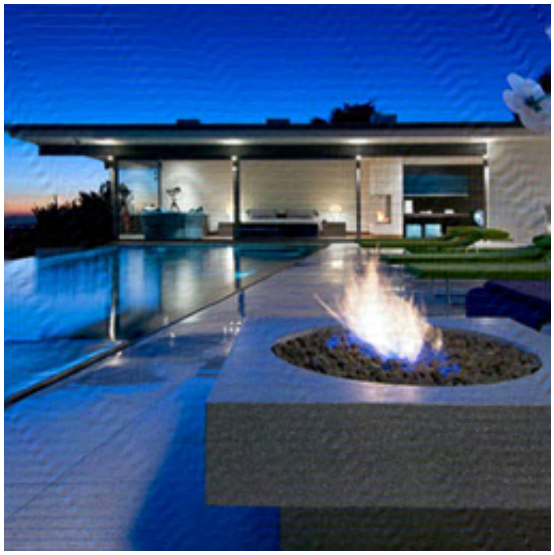


(b) $\sigma_k = 1.5$

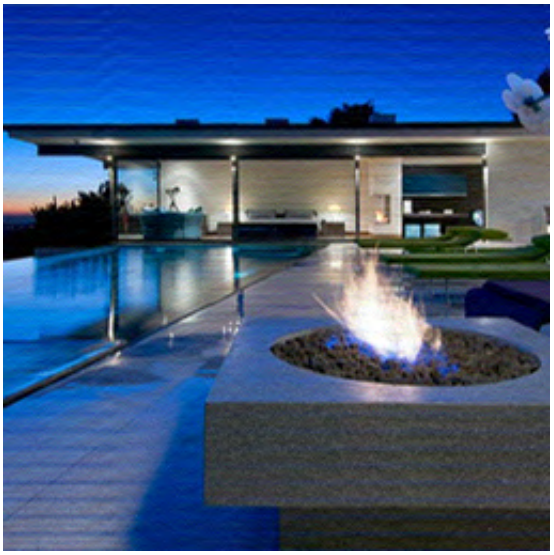
(c) $\sigma_k = 2$

(d) $\sigma_k = 2.5$

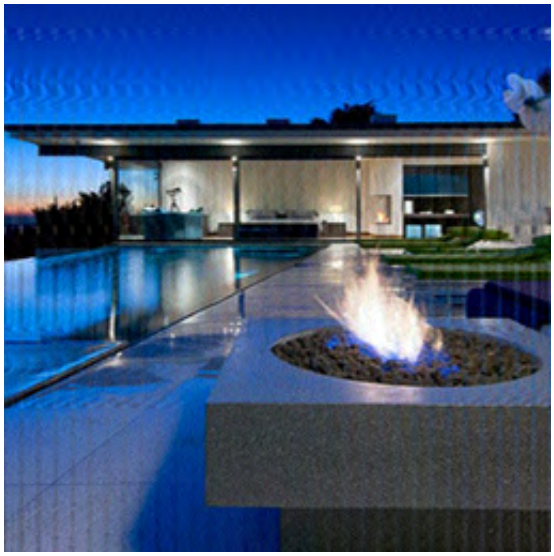
Figure 69: This figure shows 4 sample images obtained by applying the deblurring operation using Tikhonov regularization (Matlab's *deconvreg*) to images corrupted by Gaussian blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.



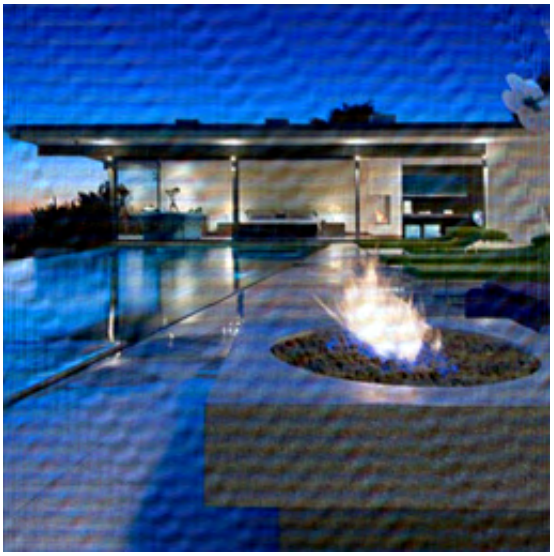
(a) $len = 6,\ dir = 120$



(b) $len = 8,\ dir = 90$



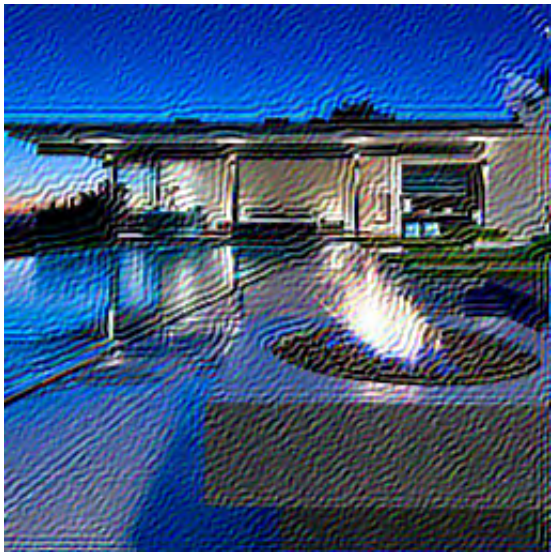
(c) $len = 8,\ dir = 180$



(d) $len = 10,\ dir = 300$

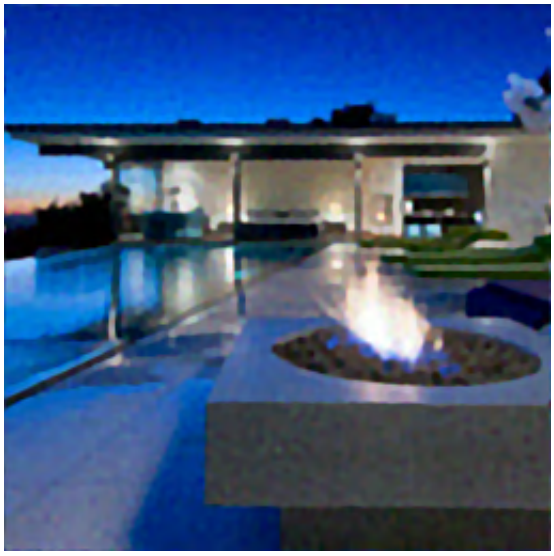
Figure 70: This figure shows 4 sample images obtained by applying the deblurring operation using Tikhonov regularization (Matlab's *deconvreg*) to images corrupted by motion blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

9-10. Joint deblurring and denoising using Chan et al.'s method [50]: Chan et al.'s method can be applied for restoring images that are corrupted by additive noise, in addition to blur. We apply this method to images corrupted by 1) Gaussian blur and additive Gaussian noise, and 2) motion blur and additive Gaussian noise. For Gaussian blur, we set the kernel size (s_k) to be a function of the standard deviation (σ_k): $s_k = 3 \times \sigma_k + 2$.
Parameters: Gaussian blur kernel standard deviation: $\sigma_k \in [0.5, 2.5]$, motion blur kernel length: $len \in [3, 25]$, motion blur kernel direction: $dir \in [0, 360]$, and additive Gaussian noise variance: $\sigma_g^2 \in [10^{-5}, 8 \times 10^{-4}]$
Relevance: The distortions are representative of the artifacts caused by a number of image restoration algorithms.

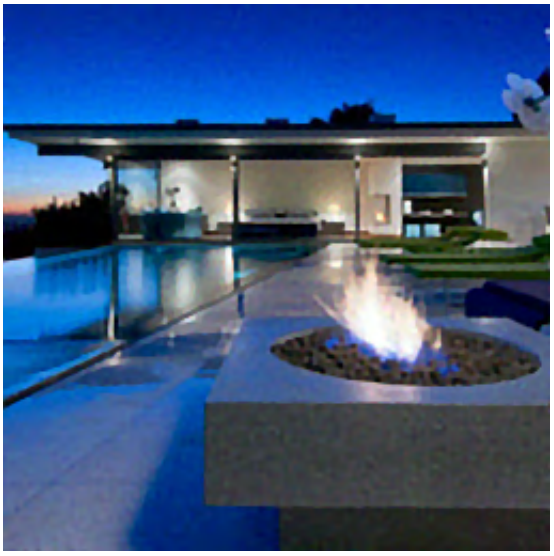


(a) $\sigma_k = 0.5,\ \sigma_g^2 = 2.1 \times 10^{-4}$

(b) $\sigma_k = 2.0,\ \sigma_g^2 = 2.1 \times 10^{-4}$



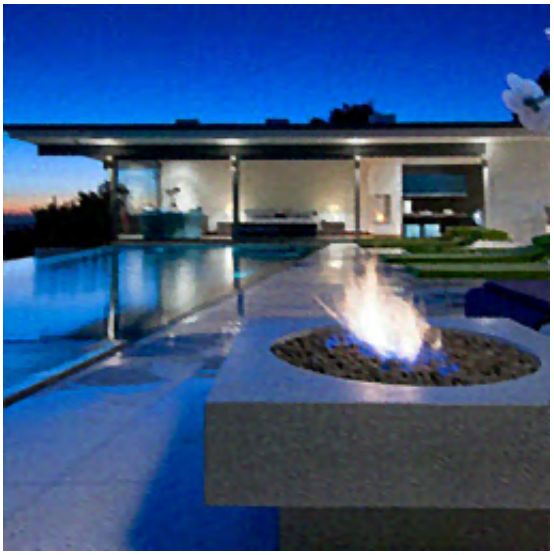
(c) $\sigma_k = 1.5,\ \sigma_g^2 = 4.1 \times 10^{-4}$



(d) $\sigma_k = 1,\ \sigma_g^2 = 2.1 \times 10^{-4}$

Figure 71: This figure shows 4 sample images obtained by applying joint deblurring and denoising using Chan et al.'s method [50] to images corrupted by additive Gaussian noise and Gaussian blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

(a) $len = 3,\ dir = 300,\ \sigma_g^2 = 2.1 \times 10^{-4}$



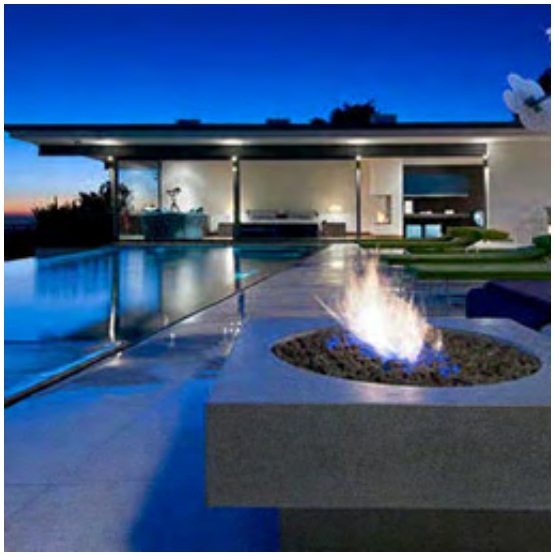
(b) $len = 4,\ dir = 330,\ \sigma_g^2 = 2.1 \times 10^{-4}$

(c) $len = 3,\ dir = 30,\ \sigma_g^2 = 6.1 \times 10^{-4}$

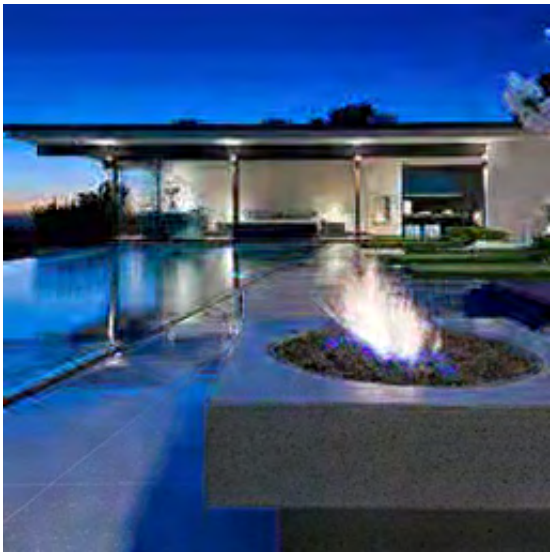
(d) $len = 10,\ dir = 240,\ \sigma_g^2 = 4.1 \times 10^{-4}$

Figure 72: This figure shows 4 sample images obtained by applying joint deblurring and denoising using Chan et al.'s method [50] to images corrupted by additive Gaussian noise and motion blur. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

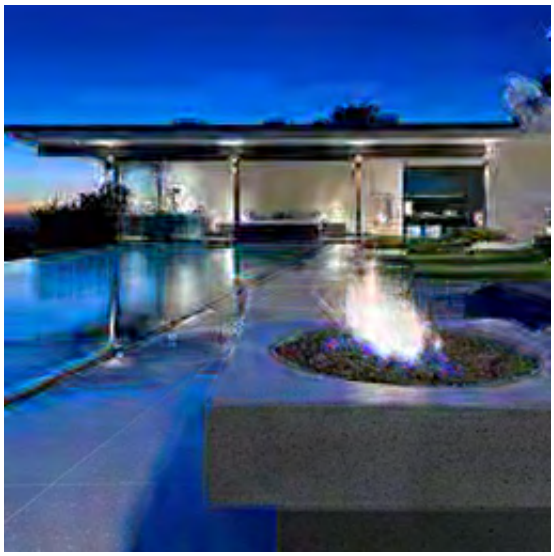
11. Comfort noise: Comfort noise is used to offset the artifacts caused by lossy compression [58]. We use the implementation of comfort noise in [4], which is described as the following. An image is first converted to the YCbCr space from the RGB space. Each color channel is lossy-compressed using ADCT [45]. Each channel is then decompressed and post-processed for block artifacts removal to yield a reconstructed channel, Y_r. Given that the original channel is Y, the corresponding channel of the distorted image is generated as follows: $Y_D = 2 \times Y_r - Y$.
Parameters: Quantization step for ADCT: $\Delta \in [10, 90]$.(The input image pixel values range from 0 to 255 in this case.)
Relevance: This distortion can occur in image compression and de-compression pipelines.



(a) $\Delta = 10$



(b) $\Delta = 35$

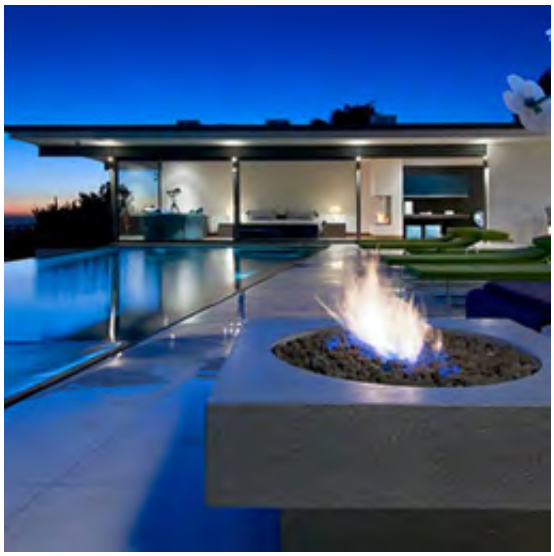


(c) $\Delta = 50$

(d) $\Delta = 85$

Figure 73: This figure shows 4 sample images corrupted by comfort noise [4]. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

12. JPEG2000 compression: Images are compressed using the JPEG2000 compression algorithm (using the *jp2* option in Matlab).
Parameters: compression ratio: $R \in [10, 100]$
Relevance: This distortion commonly occurs in image compression applications.



(a) $R = 10$

(b) $R = 30$

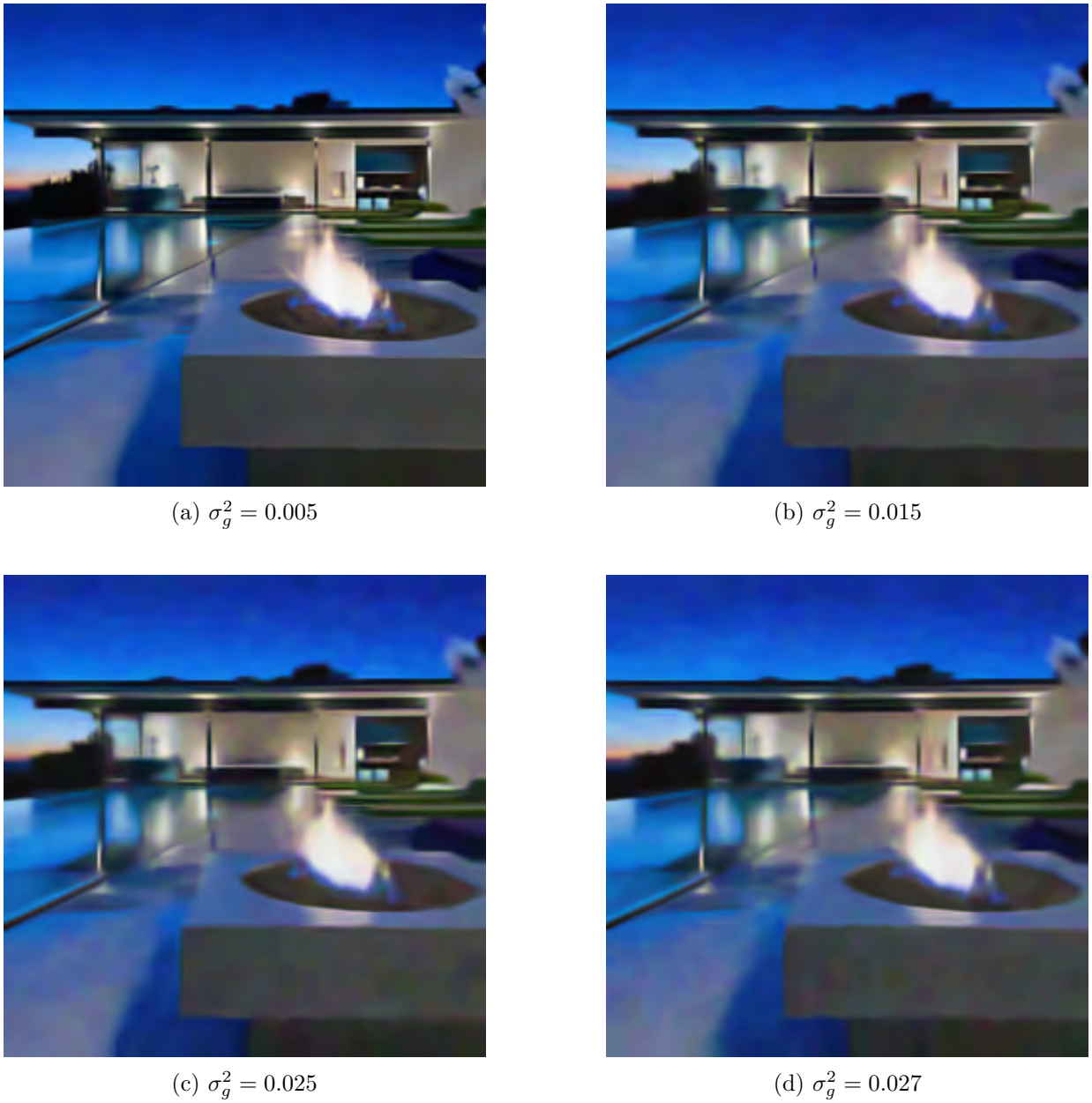
(c) $R = 60$

(d) $R = 65$

Figure 74: This figure shows 4 sample images corrupted by JPEG2000 compression. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

6.2.3 Detail loss

13-14. BM3D denoising [51]: BM3D (using default settings) has been included in the test set to denoise images corrupted by **1)** additive Gaussian noise of a constant variance, and **2)** additive Gaussian noise of a spatially-varying variance. In the spatially-varying case, similar to our other spatially-varying distortions, a Perlin noise pattern [46] is used to obtain the per-pixel noise variance. The range of values obtained from the Perlin noise pattern is scaled to lie between 0 and a specified maximum variance
Parameters: Additive Gaussian noise variance for constant variance noise: $\sigma_g^2 \in [0.01, 0.09]$, the maximum noise variance for spatially-varying noise: $\sigma_{\max}^2 \in [0.01, 0.09]$, and the maximum scale for Perlin noise generator: $s_{\max} \in [5, 8]$
Relevance: These distortions capture the artifacts caused by the popular BM3D denoising algorithm.



(a) $\sigma_g^2 = 0.005$

(b) $\sigma_g^2 = 0.015$

(c) $\sigma_g^2 = 0.025$

(d) $\sigma_g^2 = 0.027$

Figure 75: This figure shows 4 sample images obtained by applying BM3D denoising [51] to images corrupted by additive Gaussian noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

(a) $\sigma^2_{\max} = 0.02,\ s_{\max} = 7$

(b) $\sigma^2_{\max} = 0.04,\ s_{\max} = 8$

(c) $\sigma^2_{\max} = 0.06,\ s_{\max} = 6$

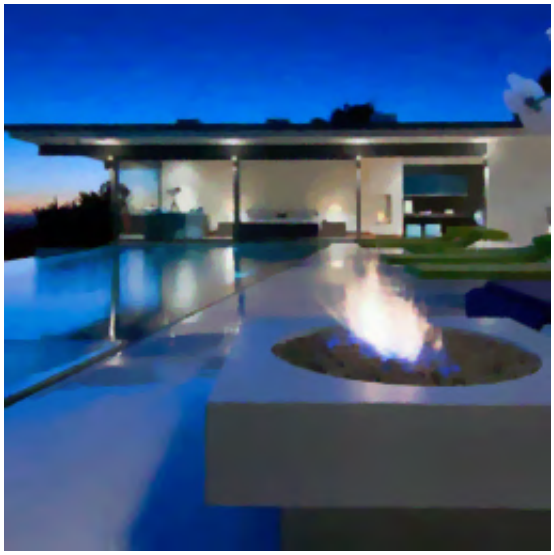
(d) $\sigma^2_{\max} = 0.08,\ s_{\max} = 8$

Figure 76: This figure shows 4 sample images obtained by applying BM3D denoising [51] to images corrupted by additive Gaussian noise with a spatially-varying variance. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

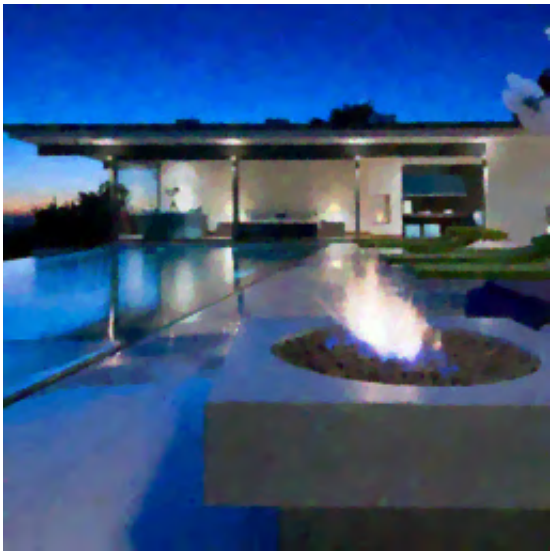
15. ROF denoising using split Bregman solver [37, 52]: ROF denoising implemented with split Bregman method is used (with default settings) to denoise an image corrupted by additive Gaussian noise. Note that the artifacts caused by this distortion are different from those caused by ROF denoising using a fixed-point iteration solver (training distortion No. 23).
Parameters: Additive Gaussian standard deviation: $\sigma_g \in [5, 30]$. (The pixel values range from 0 to 255 in this case.)
Relevance: This distortion captures the artifacts caused by the popular ROF denoising algorithm.

(a) $\sigma_g = 5$

(b) $\sigma_g = 10$



(c) $\sigma_g = 15$



(d) $\sigma_g = 24$

Figure 77: This figure shows 4 sample images obtained by applying ROF denoising (with split Bregman solver) [37,52] to images corrupted by additive Gaussian noise. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

16. Compressive sensing using Danielyan et al.'s method [53]: This is a compressive sensing technique that adopts spatial filtering as a form of regularization. It is used to reconstruct an image from its sparse samples (with default settings for the algorithm, as provided by the authors).
Parameters: Number of samples as a fraction of the rows and columns of an images: $f_s \in [0.1, 0.6]$
Relevance: This distortion occurs from the sparse sampling and reconstruction of an image.

(a) $f_s = 0.45$

(b) $f_s = 0.35$

(c) $f_s = 0.2$

(d) $f_s = 0.1$

Figure 78: This figure shows 4 sample images obtained by sparse sampling and reconstructing the reference image using Danielyan et al.'s compressive sensing algorithm [53]. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

17-20. Super-resolution [54–57]: Images are first downsampled with a resizing factor and then up-scaled using several popular super-resolution algorithms that use a variety of techniques, ranging from sparse dictionary learning to convolutional neural networks: Dong et al. [54], Peleg et al. [55], Timofte et al. [56], and Zeyde et al. [57]. In some cases, the resultant images have fewer pixels (lost from the image borders). Matlab's *padarray* function with *replicate* option is used to make sure that the size of the super-resolved image is the same as the reference image.
Parameter: The resizing factor: $u \in \{2, 3, 4, 6, 8\}$ (except for Peleg et al.'s, where $u \in \{2, 3\}$).
Relevance: These distortions are representative of the artifacts caused by a variety of super-resolution algorithms.

(a) $u = 2$

(b) $u = 4$

(c) $u = 6$

(d) $u = 8$

Figure 79: This figure shows 4 sample images obtained by applying super-resolution to downsampled versions of the reference image using Dong et al.'s method [54]. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

(a) $u = 2$

(b) $u = 3$

Figure 80: This figure shows 2 sample images obtained by applying super-resolution to downsampled versions of the reference image using Peleg et al.'s method [55]. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

(a) $u = 2$

(b) $u = 3$

(c) $u = 4$

(d) $u = 8$

Figure 81: This figure shows 4 sample images obtained by applying super-resolution to downsampled versions of the reference image using Timofte et al.'s method [56]. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

(a) $u = 2$

(b) $u = 4$

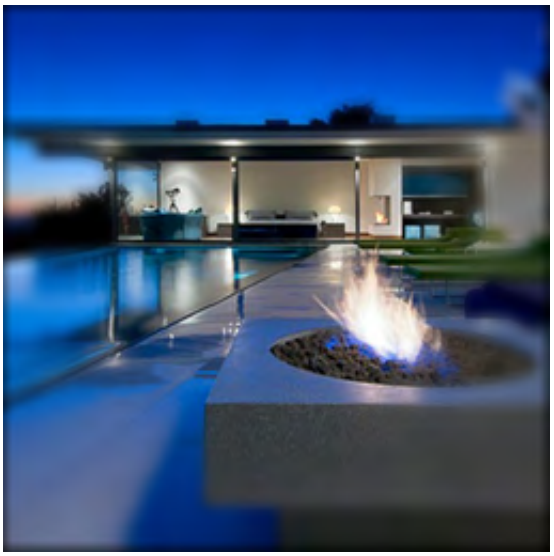
(c) $u = 6$

(d) $u = 8$

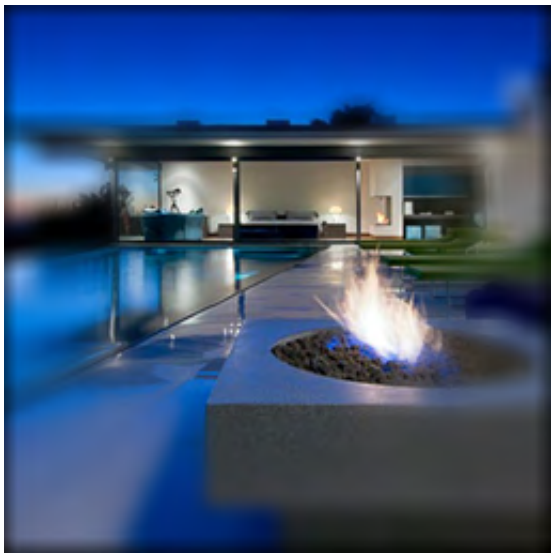
Figure 82: This figure shows 4 sample images obtained by applying super-resolution to downsampled versions of the reference image using Zeyde et al.'s method [57]. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

21. Soft focus: The distorted image is given by $I_D = W \circ I_R + (1 - W) \circ \widetilde{I}$, where I_R is the reference image, $\widetilde{I}$ is a Gaussian blurred version of I, W is a radially symmetric weight map that weighs the blurred version more as the radial distance (normalized so that the distance is 0 at the image center and close to 1 near the edges) from image center increases, and "$\circ$" denotes Hadamard product. Along a single ray from center to an image edge point, $W(p) = r_w \times (1 - d_w(p))^{g_w}$, where p is a point along the ray and $d_w(p)$ is the normalized radial distance of point p. r_w and g_w are scalar parameters that govern the fall-off of W as the distance from the origin increases.
Parameters: The Gaussian blur kernel standard deviation: $\sigma_k \in [1, 15]$ and the parameters governing the fall-off: $r_w \in [0.5, 10]$, $g_w \in [2, 20]$.(The pixel values range from 0 to 255 in this case.)
Relevance: This distortion can simulate a lens flaw causing blurred images. It can also be deliberately introduced to create artistic effects in photography.

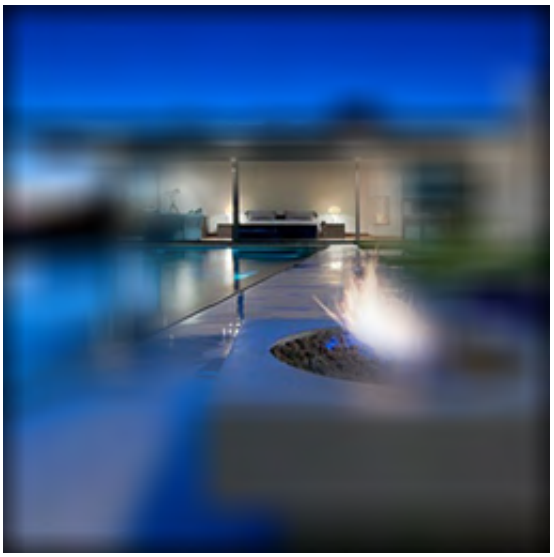
(a) $\sigma_k = 3,\ r_w = 2,\ g_w = 4$



(b) $\sigma_k = 6,\ r_w = 4,\ g_w = 5$



(c) $\sigma_k = 9,\ r_w = 5,\ g_w = 9$

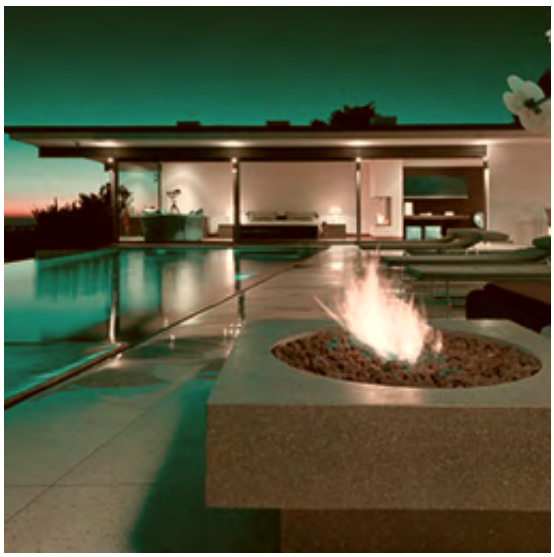


(d) $\sigma_k = 12,\ r_w = 10,\ g_w = 16$

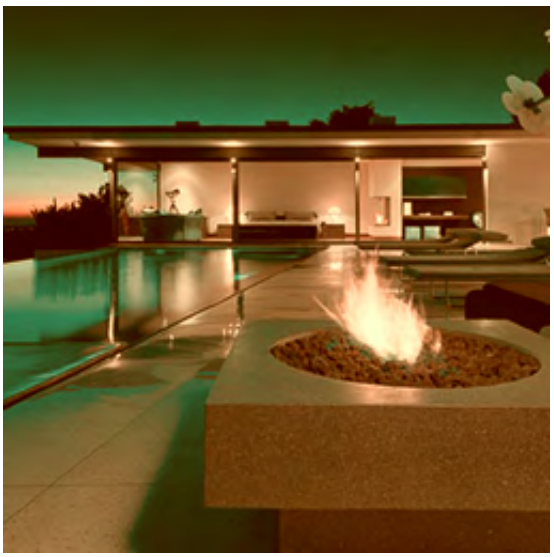
Figure 83: This figure shows 4 sample images obtained by applying the soft focus effect to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

6.2.4 Color change

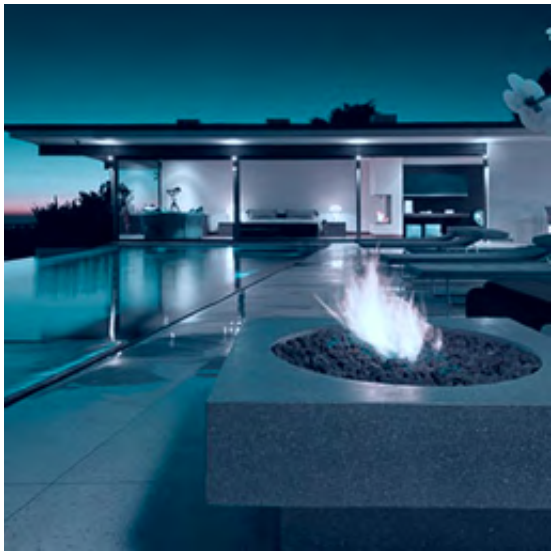
22. Color temperature change: The colors in the R and B channel are shifted by equal amounts in the opposite directions. This is as a simple approximation of change in color temperature.
Parameter: The value of color shift: $s_t \in [-50, 50]$.(The pixel values range from 0 to 255 in this case.)
Relevance: This distortion can occur from color correction algorithms. It is also a popular way to create artistic photo effects.



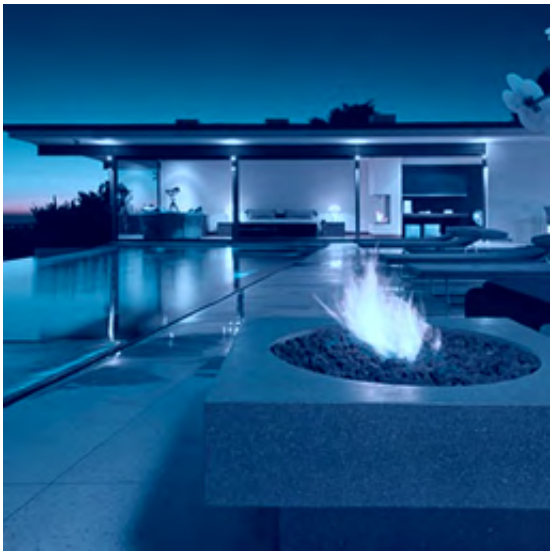
(a) $s_t = 25$



(b) $s_t = 50$



(c) $s_t = -25$



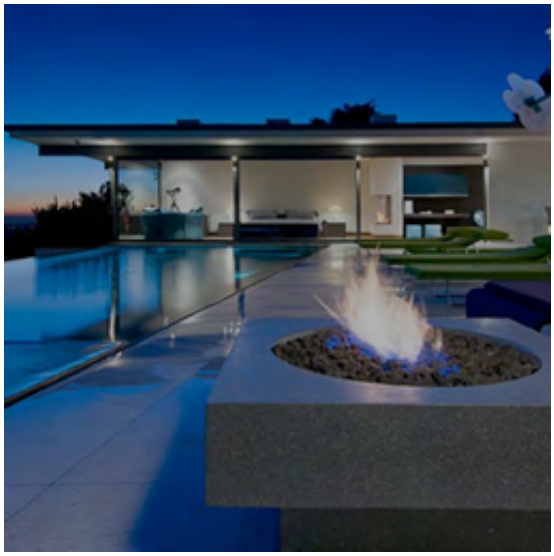
(d) $s_t = -50$

Figure 84: This figure shows 4 sample images obtained by applying color temperature change to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

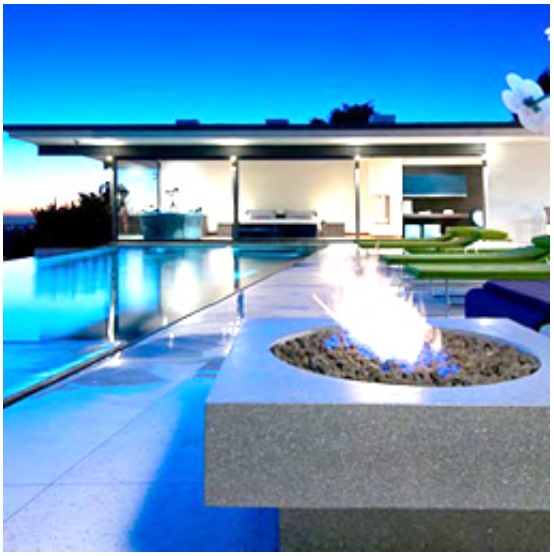
23. Log transformation: A distorted image (I_D) is obtained from the reference image (I_R) by the following transformation:

$$I_D(x, y, c) = \alpha_{\log} \times \log(1 + I_R(x, y, c)), \tag{10}$$

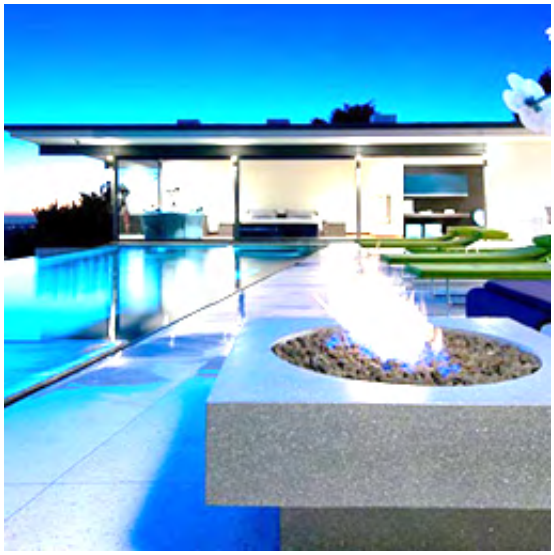
for each pixel location (x, y) and each color channel c.
Parameter: $\alpha_{\log} \in [0.4, 3]$
Relevance: This distortion captures the output of using Log transformation for global color correction, which is a commonly-used technique.



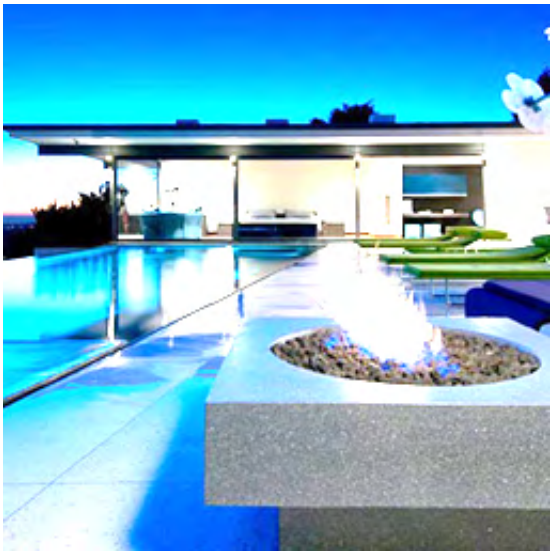
(a) $\alpha_{\log} = 1$



(b) $\alpha_{\log} = 2.2$



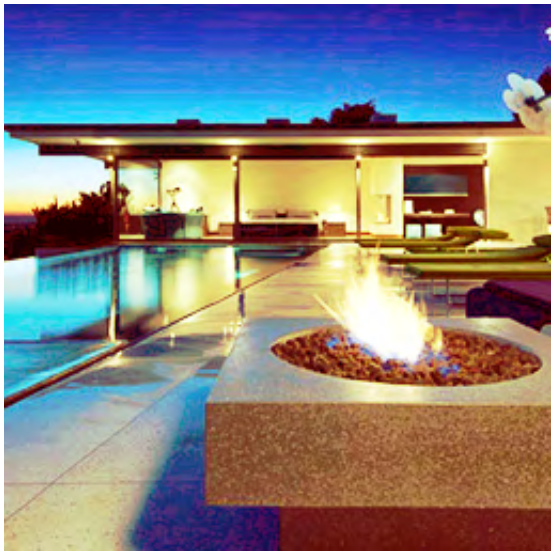
(c) $\alpha_{\log} = 2.6$



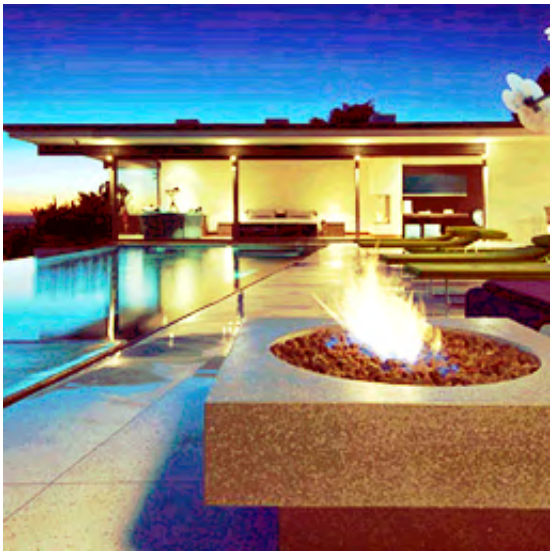
(d) $\alpha_{\log} = 2.8$

Figure 85: This figure shows 4 sample images obtained after applying log transformation to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

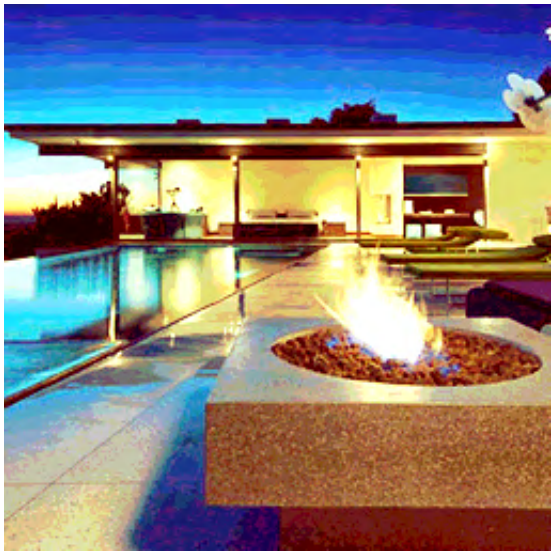
24. Histogram equalization: Histogram equalization is applied (using Matlab's *histeq*) to the R, G, and B channels of the reference image independently with a specified number of histogram bins to obtain the distorted image.
Parameter: Number of bins of the output image: $n_{\text{bin}} \in [5, 200]$.
Relevance: This distortion captures the output of using histogram equalization for global color correction, which is a commonly-used technique.



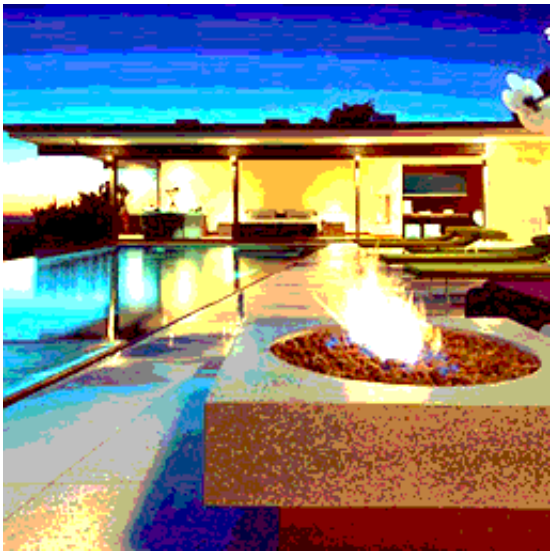
(a) $n_{\text{bin}} = 50$



(b) $n_{\text{bin}} = 25$



(c) $n_{\text{bin}} = 10$



(d) $n_{\text{bin}} = 5$

Figure 86: This figure shows 4 sample images obtained by applying per-channel histogram equalization to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

25. Vignette effect: The distorted image is given by $I_D = B \circ I_R$, where I_R is the reference image, B is a radially symmetric color attenuation map which goes from 1 to 0 as the radial distance (normalized so that the distance is 0 at the image center and close to 1 near the edges) from image center increases, and "$\circ$" denotes Hadamard product. More specifically, along a single ray from center to an image edge point, $B(p) = r_b \times (1 - d_b(p))^{g_b}$, where p is a point along the ray, $d_b(p)$ is the normalized radial distance of point p. r_b and g_b are scalar parameters that govern the fall-off of B as the distance from the origin increases.
Parameters: The parameters governing the fall-off: $r_b \in [1, 7]$, $g_b \in [0.5, 2]$. (The pixel values range from 0 to 255 in this case.)
Relevance: This distortion simulates the vignetting of pictures coming from DSLRs and artistic image processing.

(a) $r_b = 6,\ g_b = 0.8$

(b) $r_b = 2,\ g_b = 0.8$

(c) $r_b = 4,\ g_b = 1.5$

(d) $r_b = 1,\ g_b = 0.5$

Figure 87: This figure shows 4 sample images obtained by applying the vignette effect to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

6.2.5 Geometric transformations

26. Vertical image stretch/shrink: An image is stretched/shrunk vertically by increasing/decreasing the number of rows in the image and using bicubic interpolation (using *imresize* function in Matlab). The resulting image is cropped at the center or hole-filled using GMIC [47] near the edges to match the size of the reference image.
Parameter: Resizing factor in the vertical direction: $r_v \in [0.8, 1.5]$
Relevance: This distortion can occur during panoramic image stitching, 3D reconstruction, and image morphing.

(a) $r_v = 0.8$

(b) $r_v = 0.9$

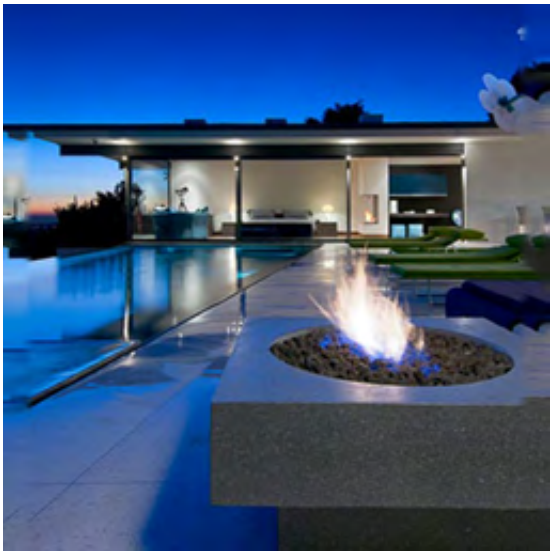
(c) $r_v = 1.1$

(d) $r_v = 1.3$

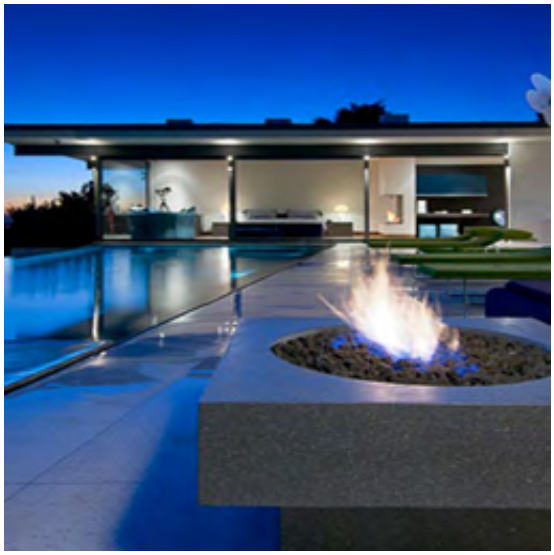
Figure 88: This figure shows 4 sample images obtained by applying vertical stretching/shrinking to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

27. Horizontal image stretch/shrink: An image is stretched/shrunk horizontally by increasing/decreasing the number of columns in the image and using bicubic interpolation (using *imresize* function in Matlab). The resulting image is cropped at the center or hole-filled using GMIC [47] near the edges to match the size of the reference image.
Parameter: Resizing factor in the vertical direction: $r_h \in [0.8, 1.5]$
Relevance: This distortion can occur during panoramic image stitching, 3D reconstruction and image morphing.

(a) $r_h = 0.8$



(b) $r_h = 0.9$



(c) $r_h = 1.1$

(d) $r_h = 1.3$

Figure 89: This figure shows 4 sample images obtained by applying horizontal stretching/shrinking to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

28. Swirl transformation: The image is resampled on a new coordinate space (x', y'), given the original coordinate space (x, y), as follows:

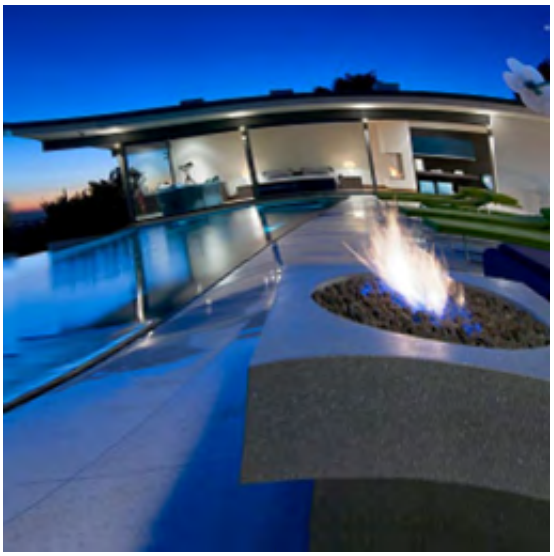
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta(x,y) & \sin\theta(x,y) \\ -\sin\theta(x,y) & \cos\theta(x,y) \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}. \tag{11}$$

$\theta(x, y)$ is a rotation angle map computed as follows:

$$\theta(x,y) = \theta_0 \times (1 - \frac{r_\theta(x,y)}{\max\limits_{x,y} r_\theta(x,y)}), \tag{12}$$

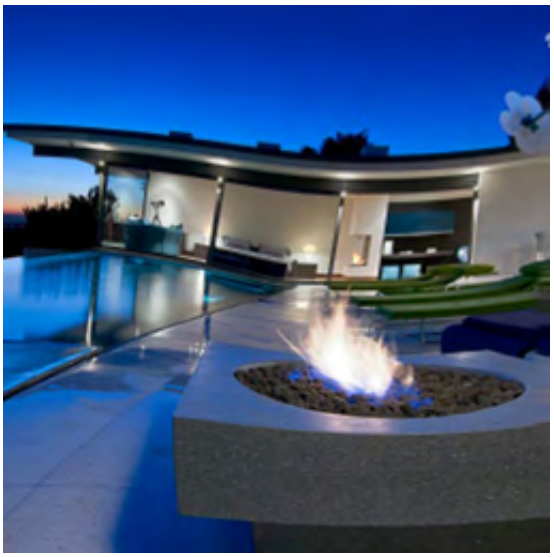
where $r_\theta(x,y) = \sqrt{(x-x_0)^2 + (y-y_0)^2}$. The resulting image is hole-filled near the edges using GMIC [47].
Parameters: $x_0 \in [-0.25 \times W_I, 0.25 \times W_I]$, $y_0 \in [-0.25 \times H_I, 0.25 \times H_I]$, and $\theta_0 \in [-20, 20]$, where W_I and H_I are the height and width of the image, respectively.
Relevance: This distortion can occur from image morphing and artistic geometric image transformations.

(a) $x_0 = -0.2 \times W_I,\ y_0 = -0.25 \times H_I,\ \theta_0 = -11$



(b) $x_0 = -0.1 \times W_I,\ y_0 = 0.1 \times H_I,\ \theta_0 = 18$

(c) $x_0 = 0,\ y_0 = -0.25 \times H_I,\ \theta_0 = -19$



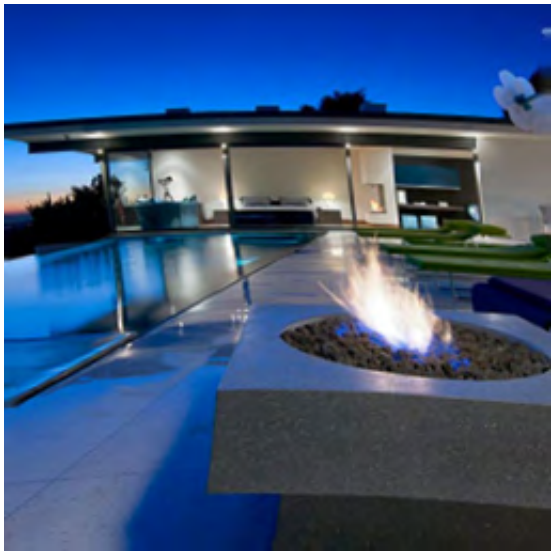
(d) $x_0 = 0.1 \times W_I,\ y_0 = 0.1 \times H_I,\ \theta_0 = -19$

Figure 90: This figure shows 4 sample images obtained by applying the swirl transformation to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

29. Wave transformation: The image is resampled on a new coordinate space (x', y'), given the original coordinate space (x, y), as follows:

$$x' = x + a_x \times \sin(\mu_x \times \bar{y}) \text{ and } x' = y + a_y \times \sin(\mu_y \times \bar{x}). \tag{13}$$

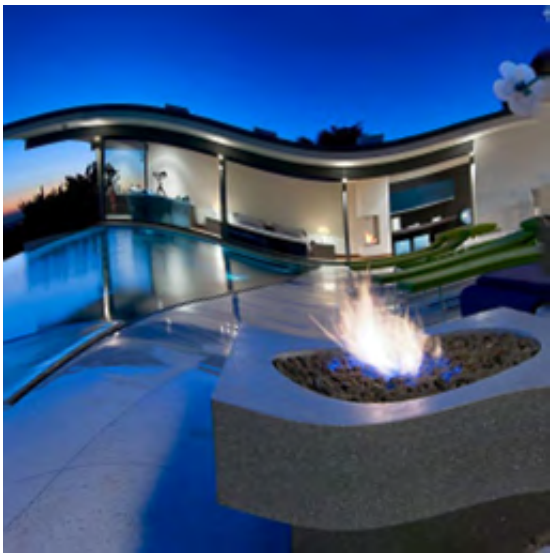
where $\bar{y} = y/H_I$ and $\bar{x} = x/W_I$, where H_I and W_I are the height and width of the image, respectively. The resulting image is hole-filled near the edges using GMIC [47].
Parameters: $a_x \in [5, 10]$, $a_y \in [5, 10]$, $\mu_x \in [0.3 \times 2\pi, 1.2 \times 2\pi]$, and $\mu_y \in [0.3 \times 2\pi, 1.2 \times 2\pi]$
Relevance: This distortion can occur from image morphing and artistic geometric image transformations.



(a) $a_x = 5,\ a_y = 6,\ \mu_x = 1.1 \times 2\pi,\ \mu_y = 0.5 \times 2\pi$

(b) $a_x = 9,\ a_y = 10,\ \mu_x = 0.8 \times 2\pi,\ \mu_y = 0.4 \times 2\pi$

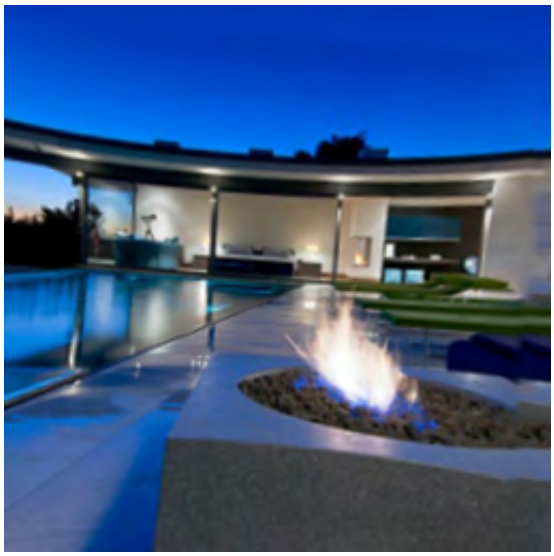
(c) $a_x = 9,\ a_y = 10,\ \mu_x = 0.3 \times 2\pi,\ \mu_y = 2\pi$



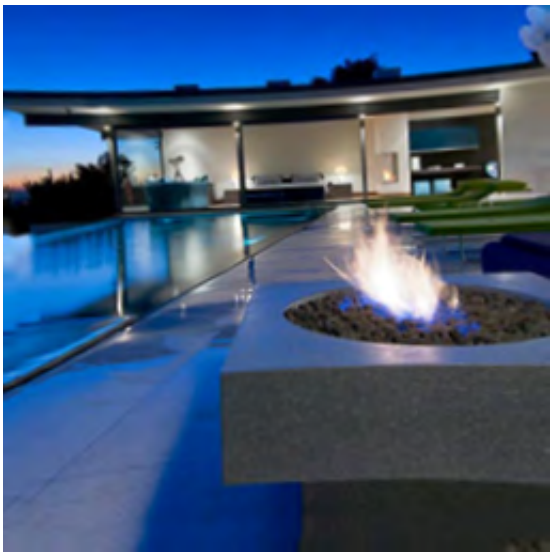
(d) $a_x = 6,\ a_y = 10,\ \mu_x = 0.8 \times 2\pi,\ \mu_y = 1.2 \times 2\pi$

Figure 91: This figure shows 4 sample images obtained by applying the wave transformation to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

30. Image shift and rotation and radial distortion: An image is spatially shifted, rotated using a 2D rotation matrix and distorted using either pincushion or barrel distortion. This distortion is a combination of training distortions No. 36-40 (see Sec. 6.1.6 of this file) to create new visual effects.
Parameters: Horizontal spatial shift: $s_h \in [-10, 10]$, vertical spatial shift: $s_v \in [-10, 10]$, the 2D rotation angle: $\theta \in [-9, 9]$, and the radial transformation coefficient $a \in [-7 \times 10^{-6}, 5 \times 10^{-6}]$
Relevance: This is a common distortion from 3D image processing applications (e.g., panoramic stitching) and lens distortion correction.



(a) $s_h = -4,\ s_v = 8,\ \theta = 4,\ a = -6.5 \times 10^{-6}$



(b) $s_h = -2,\ s_v = -10,\ \theta = -3,\ a = -3.5 \times 10^{-6}$

(c) $s_h = 10,\ s_v = 4,\ \theta = 8,\ a = -6 \times 10^{-6}$

(d) $s_h = -7,\ s_v = -6,\ \theta = -8,\ a = 5 \times 10^{-6}$

Figure 92: This figure shows 4 sample images obtained by applying a combination of spatial shift, rotation and radial distortion to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

31. Radial distortion using 2nd order polynomial function: This distortion is formulated as a general 2^{nd}-degree radially symmetric polynomial in the polar coordinate space:

$$r' = a_0 + a_1 \times r + a_2 \times r^2 \text{ and } \theta' = \theta, \tag{14}$$

where (r, θ) are the original polar coordinates computed about an arbitrary location (x_0, y_0) of the image and (r', θ') are the transformed polar coordinates. The resulting image is hole-filled near the edges using GMIC [47].
Parameters: $a_0 \in [5, 50]$, $a_1 \in [0.5, 1]$, and $a_2 \in [-0.001, 0.001]$
Relevance: This distortion is used to model lens distortions.

(a) $a_0 = 7.93,\ a_1 = 0.94,\ a_2 = -5.2 \times 10^{-4}$

(b) $a_0 = 9.03,\ a_1 = 0.94,\ a_2 = -6.6 \times 10^{-4}$

(c) $a_0 = 19,\ a_1 = 0.875,\ a_2 = -2.6 \times 10^{-4}$

(d) $a_0 = 42,\ a_1 = 0.71,\ a_2 = 4 \times 10^{-5}$

Figure 93: This figure shows 4 sample images obtained by applying the radial distortion (with a general 2^{nd} order polynomial) to the reference image. The parameter settings used to generate each of the 4 images are stated in the caption of each sub-figure.

References

[1] N. Ponomarenko, V. Lukin, A. Zelensky, and K. Egiazarian. TID2008-A database for evaluation of full-reference visual quality assessment metrics. *Advances of Modern Radioelectronics*, 10(4):30–45, 2009.

[2] H. R. Sheikh, M. F. Sabir, and A. C. Bovik. A statistical evaluation of recent full reference image quality assessment algorithms. *IEEE Transactions on Image Processing*, 15(11):3440–3451, 2006.

[3] E. C. Larson and D. M. Chandler. Most apparent distortion: Full-reference image quality assessment and the role of strategy. *Journal of Electronic Imaging*, 19(1):011006–011006, 2010.

[4] N. Ponomarenko, L. Jin, O. Ieremeiev, V. Lukin, K. Egiazarian, J. Astola, B. Vozel, K. Chehdi, M. Carli, F. Battisti, et al. Image database TID2013: Peculiarities, results and perspectives. *Signal Processing: Image Communication*, 30:57–77, 2015.

[5] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[6] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1398–1402, 2003.

[7] N. Ponomarenko, O. Ieremeiev, V. Lukin, K. Egiazarian, and M. Carli. Modified image visual quality metrics for contrast change and mean shift accounting. In *Proceedings of the International Conference on the Experience of Designing and Application of CAD Systems in Microelectronics*, pages 305–311, 2011.

[8] L. Zhang, L. Zhang, X. Mou, and D. Zhang. FSIM: A feature similarity index for image quality assessment. *IEEE Transactions on Image Processing*, 20(8):2378–2386, 2011.

[9] H. Chang, H. Yang, Y. Gan, and M. Wang. Sparse feature fidelity for perceptual image quality assessment. *IEEE Transactions on Image Processing*, 22(10):4007–4018, 2013.

[10] W. Xue, L. Zhang, X. Mou, and A. C. Bovik. Gradient magnitude similarity deviation: A highly efficient perceptual image quality index. *IEEE Transactions on Image Processing*, 23(2):684–695, 2014.

[11] L. Zhang, Y. Shen, and H. Li. VSI: A visual saliency-induced index for perceptual image quality assessment. *IEEE Transactions on Image Processing*, 23(10):4270–4281, 2014.

[12] S. Bae and M. Kim. A novel image quality assessment with globally and locally consilient visual quality perception. *IEEE Transactions on Image Processing*, 25(5):2392–2406, 2016.

[13] S. Pei and . Chen. Image quality assessment using human visual DOG model fused with random forest. *IEEE Transactions on Image Processing*, 24(11):3282–3292, 2015.

[14] V. Lukin, N. Ponomarenko, O. Ieremeiev, K. Egiazarian, and J. Astola. Combining full-reference image visual quality metrics by neural network. In *SPIE/IS&T Electronic Imaging*, 2015.

[15] S. Bosse, D. Maniry, K. R. Müller, T. Wiegand, and W. Samek. Full-reference image quality assessment using neural networks. *Proceedings of the International Workshop on Video Processing and Quality Metrics*, 2016.

[16] Jongyoo Kim and Sanghoon Lee. Deep learning of human visual sensitivity in image quality assessment framework. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[17] N. Damera-Venkata, T. D. Kite, W. S. Geisler, B. L. Evans, and A. C. Bovik. Image quality assessment based on a degradation model. *IEEE Transactions on Image Processing*, 9(4):636–650, 2000.

[18] H. R. Sheikh, A. C. Bovik, and G. De Veciana. An information fidelity criterion for image quality assessment using natural scene statistics. *IEEE Transactions on image processing*, 14(12):2117–2128, 2005.

[19] H. R. Sheikh and A. C. Bovik. Image information and visual quality. *IEEE Transactions on Image Processing*, 15(2):430–444, 2006.

[20] D. M. Chandler and S. S. Hemami. VSNR: A wavelet-based visual signal-to-noise ratio for natural images. *IEEE Transactions on Image Processing*, 16(9):2284–2298, 2007.

[21] L. Zhang, L. Zhang, and X. Mou. RFSIM: A feature based image quality assessment metric using Riesz transforms. In *Proceedings of the IEEE International Conference on Image Processing*, pages 321–324, 2010.

[22] A. Liu, W. Lin, and M. Narwaria. Image quality assessment based on gradient similarity. *IEEE Transactions on Image Processing*, 21(4):1500–1512, 2012.

[23] L. Zhang and H. Li. SR-SIM: A fast and high performance IQA index based on spectral residual. In *Proceedings of the IEEE International Conference on Image Processing*, pages 1473–1476, 2012.

[24] H. Z. Nafchi, A. Shahkolaei, R. Hedjam, and M. Cheriet. Mean deviation similarity index: Efficient and reliable full-reference image quality evaluator. *IEEE Access*, 4:5579–5590, 2016.

[25] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] K. Tsukida and M. R. Gupta. How to analyze paired comparison data. Technical report, University of Washington, Seattle, Deptartment of Electrical Engineering, 2011.

[27] R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[28] P. O'Donovan, J. Lībeks, A. Agarwala, and A. Hertzmann. Exploratory font selection using crowdsourced attributes. *ACM Transactions on Graphics*, 33(4):92, 2014.

[29] P. Le Callet and F. Autrusseau. Subjective quality assessment IRCCyN/IVC database, 2005.

[30] S. Tourancheau, F. Autrusseau, Z.M. P. Sazzad, and Y. Horita. Impact of subjective dataset on the performance of image quality metrics. In *Proceedings of the IEEE International Conference on Image Processing*, pages 365–368, 2008.

[31] U. Engelke, H. Zepernick, and T. M. Kusuma. Subjective quality assessment for wireless image communication: The wireless imaging quality database. In *International Workshop on Video Processing and Quality Metrics for Consumer Electronics*, 2010.

[32] A. Zarić, N. Tatalović, N. Brajković, H. Hlevnjak, M. Lončarić, E. Dumić, and S. Grgić. VCL@FER image quality assessment database. *AUTOMATIKA*, 53(4):344–354, 2012.

[33] X. Liu, M. Pedersen, and J. Y. Hardeberg. CID: IQ–A new image quality database. In *Proceedings of the International Conference on Image and Signal Processing*, pages 193–202, 2014.

[34] L. B. Lucy. An iterative technique for the rectification of observed distributions. *The Astronomical Journal*, 79:745, 1974.

[35] W. H. Richardson. Bayesian-based iterative method of image restoration. *JOSA*, 62(1):55–59, 1972.

[36] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[37] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.

[38] Z. Wang, D. Liu, J. Yang, W. Han, and T. Huang. Deep networks for image super-resolution with sparse prior. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 370–378, 2015.

[39] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007.

[40] N. Ponomarenko, F. Silvestri, K. Egiazarian, M. Carli, J. Astola, and V. Lukin. On between-coefficient contrast masking of DCT basis functions. In *Proceedings of the International Workshop on Video Processing and Quality Metrics*, volume 4, 2007.

[41] I. J. Cox, M. L. Miller, J. A. Bloom, and C. Honsinger. *Digital Watermarking*. Springer, 2002.

[42] R. Chaturvedi, A. Sharma, N. Hemrajani, and D. Goyal. Analysis of robust watermarking technique using mid-band DCT domain for different image formats. *International Journal of Scientific and Research Publications*, 2(3):1–4, 2012.

[43] A. Parnami, A. Gupta, and G. Parnami. A robust DCT based digital image watermarking using random mid-band coefficient exchange scheme for gray scale images. *International Journal of Computer Applications*, 100(8), 2014.

[44] C. Yang, L. Zhang, H. Lu, X. Ruan, and M. Yang. Saliency detection via graph-based manifold ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3166–3173, 2013.

[45] N. Ponomarenko, V. Lukin, K. Egiazarian, and J. Astola. ADCT: a new high quality DCT based coder for lossy image compression. *CD ROM Proceedings of LNLA*, 6, 2008.

[46] K. Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

[47] Greyc's magic for image computing. `http://gmic.eu/`.

[48] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 2(1):1–104, 2006.

[49] C. Barnes, E. Shechtman, D. B. Goldman, and A. Finkelstein. The generalized patchmatch correspondence algorithm. In *European Conference on Computer Vision*, pages 29–43, 2010.

[50] S. H. Chan, R. Khoshabeh, K. B. Gibson, P. E. Gill, and T. Q. Nguyen. An augmented Lagrangian method for total variation video restoration. *IEEE Transactions on Image Processing*, 20(11):3097–3111, 2011.

[51] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007.

[52] P. Getreuer. Rudin-Osher-Fatemi total variation denoising using split Bregman. *Image Processing On Line*, 2:74–95, 2012.

[53] A. Danielyan, A. Foi, V. Katkovnik, K. Egiazarian, and P. Milanfar. Spatially adaptive filtering as regularization in inverse imaging: Compressive sensing super-resolution and upsampling. *Super-Resolution Imaging*, pages 123–154, 2010.

[54] C. Dong, C. Loy, K. He, and X. Tang. Learning a deep convolutional network for image super-resolution. In *Proceedings of the European Conference on Computer Vision*, pages 184–199, 2014.

[55] T. Peleg and M. Elad. A statistical prediction model based on sparse representations for single image super-resolution. *IEEE Transactions on Image Processing*, 23(6):2569–2582, 2014.

[56] R. Timofte, V. De Smet, and L. Van Gool. A+: Adjusted anchored neighborhood regression for fast super-resolution. In *Proceedings of the Asian Conference on Computer Vision*, pages 111–126, 2014.

[57] R. Zeyde, M. Elad, and M. Protter. On single image scale-up using sparse-representations. In *Proceedings of the International Conference on Curves and Surfaces*, pages 711–730, 2010.

[58] J. M. Boyce and A. M. Tourapis. Comfort noise for compressed video. In *Proceedings of the International Conference on Consumer Electronics*, pages 323–324, 2005.