Image Inpainting for Irregular Holes Using Partial Convolutions

Guilin Liu Fitsum A. Reda Kevin J. Shih Ting-Chun Wang
Andrew Tao Bryan Catanzaro

NVIDIA Corporation

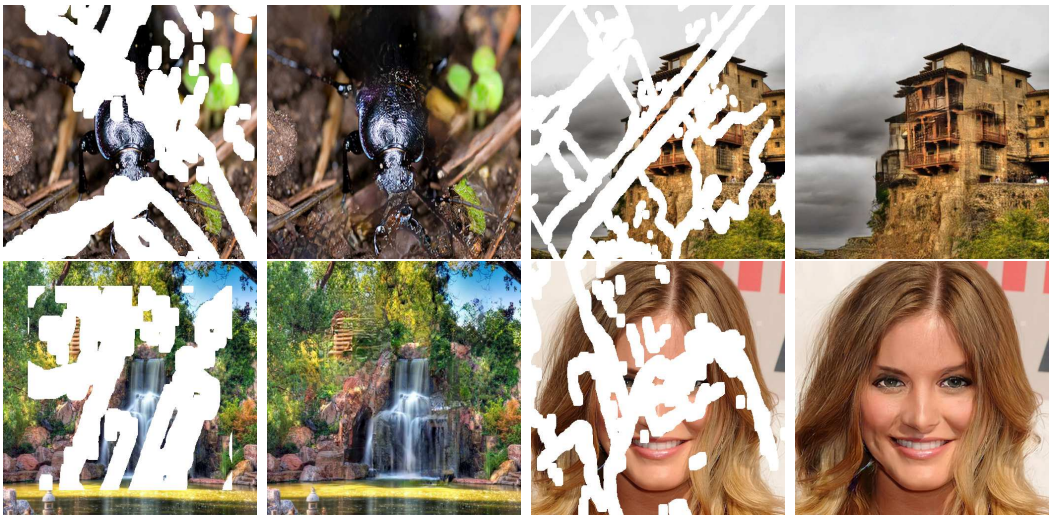


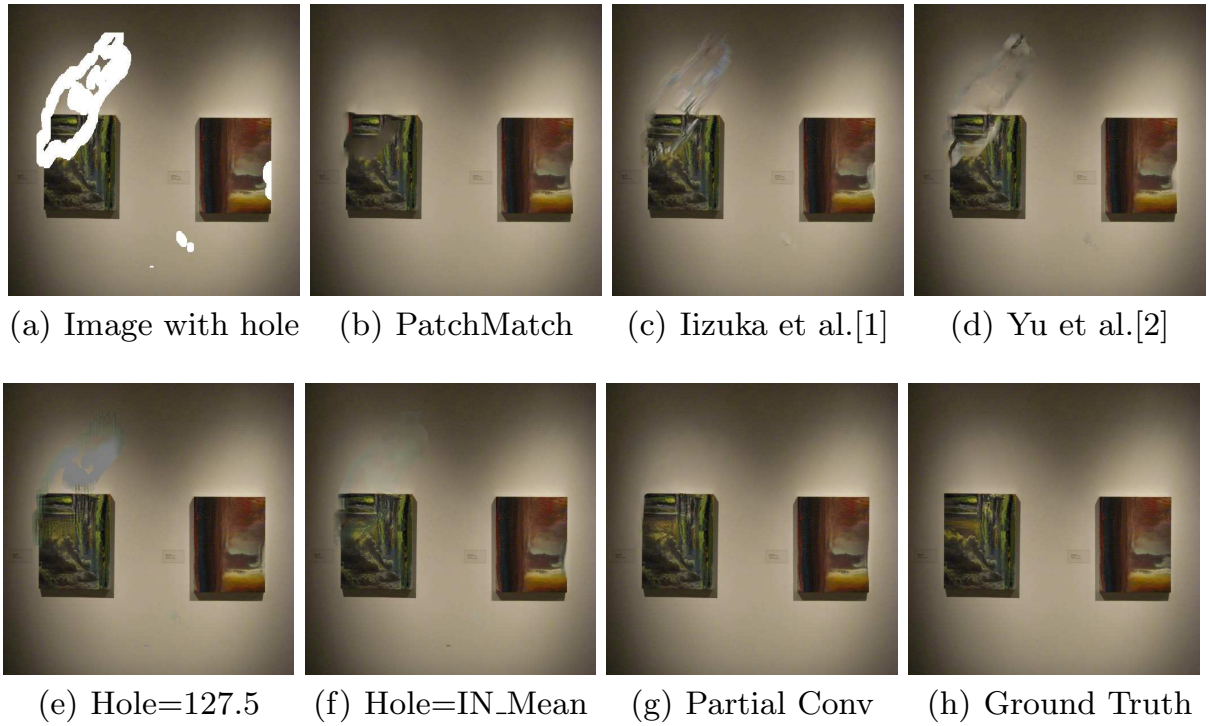
Fig. 1. Masked images and corresponding inpainted results using our partial-convolution based network.

Abstract. Existing deep learning based image inpainting methods use a standard convolutional network over the corrupted image, using convolutional filter responses conditioned on both valid pixels as well as the substitute values in the masked holes (typically the mean value). This often leads to artifacts such as color discrepancy and blurriness. Post-processing is usually used to reduce such artifacts, but are expensive and may fail. We propose the use of partial convolutions, where the convolution is masked and renormalized to be conditioned on only valid pixels. We further include a mechanism to automatically generate an updated mask for the next layer as part of the forward pass. Our model outperforms other methods for irregular masks. We show qualitative and quantitative comparisons with other methods to validate our approach.

Keywords: Partial Convolution, Image Inpainting

1 Introduction

Image inpainting, the task of filling in holes in an image, can be used in many applications. For example, it can be used in image editing to remove unwanted



(a) Image with hole (b) PatchMatch (c) Iizuka et al.[1] (d) Yu et al.[2]

(e) Hole=127.5 (f) Hole=IN_Mean (g) Partial Conv (h) Ground Truth

Fig. 2. From left to right, top to bottom: 2(a): image with hole. 2(b): inpainted result of PatchMatch[3]. 2(c): inpainted result of Iizuka et al.[1]. 2(d): Yu et al.[2]. 2(e) and 2(f) are using the same network architecture as Section 3.2 but using typical convolutional network, 2(e) uses the pixel value 127.5 to initialize the holes. 2(f) uses the mean ImageNet pixel value. 2(g): our *partial convolution* based results which are agnostic to hole values.

image content, while filling in the resulting space with plausible imagery. Previous deep learning approaches have focused on rectangular regions located around the center of the image, and often rely on expensive post-processing. The goal of this work is to propose a model for image inpainting that operates robustly on irregular hole patterns (see Fig. 1), and produces semantically meaningful predictions that incorporate smoothly with the rest of the image without the need for any additional post-processing or blending operation.

Recent image inpainting approaches that do not use deep learning use image statistics of the remaining image to fill in the hole. PatchMatch [3], one of the state-of-the-art methods, iteratively searches for the best fitting patches to fill in the holes. While this approach generally produces smooth results, it is limited by the available image statistics and has no concept of visual semantics. For example, in Figure 2(b), PatchMatch was able to smoothly fill in the missing components of the painting using image patches from the surrounding shadow and wall, but a semantically-aware approach would make use of patches from the painting instead.

Deep neural networks learn semantic priors and meaningful hidden representations in an end-to-end fashion, which have been used for recent image inpainting efforts. These networks employ convolutional filters on images, replacing the removed content with a fixed value. As a result, these approaches suffer from dependence on the initial hole values, which often manifests itself as lack of

texture in the hole regions, obvious color contrasts, or artificial edge responses surrounding the hole. Examples using a U-Net architecture with typical convolutional layers with various hole value initialization can be seen in Figure 2(e) and 2(f). (For both, the training and testing share the same initalization scheme).

Conditioning the output on the hole values ultimately results in various types of visual artifacts that necessitate expensive post-processing. For example, Iizuka et al. [1] uses fast marching [4] and Poisson image blending [5], while Yu et al. [2] employ a following-up refinement network to refine their raw network predictions. However, these refinement cannot resolve all the artifacts shown as 2(c) and 2(d). Our work aims to achieve well-incorporated hole predictions independent of the hole initialization values and without any additional post-processing.

Another limitation of many recent approaches is the focus on rectangular shaped holes, often assumed to be center in the image. We find these limitations may lead to overfitting to the rectangular holes, and ultimately limit the utility of these models in application. Pathak et al. [6] and Yang et al. [7] assume 64×64 square holes at the center of a 128×128 image. Iizuka et al. [1] and Yu et al. [2] remove the centered hole assumption and can handle irregular shaped holes, but do not perform an extensive quantitative analysis on a large number of images with irregular masks (51 test images in [8]). In order to focus on the more practical irregular hole use case, we collect a large benchmark of images with irregular masks of varying sizes. In our analysis, we look at the effects of not just the size of the hole, but also whether the holes are in contact with the image border.

To properly handle irregular masks, we propose the use of a *Partial Convolutional Layer*, comprising a masked and re-normalized convolution operation followed by a mask-update step. The concept of a masked and re-normalized convolution is also referred to as segmentation-aware convolutions in [9] for the image segmentation task, however they did not make modifications to the input mask. Our use of partial convolutions is such that given a binary mask our convolutional results depend only on the non-hole regions at every layer. Our main extension is the automatic mask update step, which removes any masking where the partial convolution was able to operate on an unmasked value. Given sufficient layers of successive updates, even the largest masked holes will eventually shrink away, leaving only valid responses in the feature map. The partial convolutional layer ultimately makes our model agnostic to placeholder hole values.

In summary, we make the following contributions:

- we propose the the use of *partial convolutions* with an *automatic mask update step* for achieving state-of-the-art on image inpainting.
- while previous works fail to achieve good inpainting results with skip links in a U-Net [10] with typical convolutions, we demonstrate that substituting convolutional layers with partial convolutions and mask updates can achieve state-of-the-art inpainting results.
- to the best of our knowledge, we are the first to demonstrate the efficacy of training image-inpainting models on irregularly shaped holes.

– we propose a large irregular mask dataset, which will be released to public to facilitate future efforts in training and evaluating inpainting models.

2 Related Work

Non-learning approaches to image inpainting rely on propagating appearance information from neighboring pixels to the target region using some mechanisms like distance field[11, 12, 4]. However, these methods can only handle narrow holes, where the color and texture variance is small. Big holes may result in over-smoothing or artifacts resembling Voronoi regions such as in [4]. Patch-based methods such as [13, 14] operate by searching for relevant patches from the image's non-hole regions or other source images in an iterative fashion. However, these steps often come at a large computation cost such as in [15]. PatchMatch [3] speeds it up by proposing a faster similar patch searching algorithm. However, these approaches are still not fast enough for real-time applications and cannot make semantically aware patch selections.

Deep learning based methods typically initialize the holes with some constant placeholder values e.g. the mean pixel value of ImageNet [16], which is then passed through a convolutional network. Due to the resulting artifacts, post-processing is often used to ameliorate the effects of conditioning on the placeholder values. Content Encoders [6] first embed the 128×128 image with 64×64 center hole into low dimensional feature space and then decode the feature to a 64x64 image. Yang et al. [7] takes the result from Content Encoders as input and then propagates the texture information from non-hole regions to fill the hole regions as postprocessing. Song et al. [17] uses a refinement network in which a blurry initial hole-filling result is used as the input, then iteratively replaced with patches from the closest non-hole regions in the feature space. Iizuka et al. [1] extends Content Encoders by defining both global and local discriminators, then applies Poisson blending as a post-process. Following [1], Yu et al. [2] replaced the post-processing with a refinement network powered by the contextual attention layers.

Amongst the deep learning approaches, several other efforts also ignore the mask placeholder values. In Yeh et al. [18], searches for the closest encoding to the corrupted image in a latent space, which is then used to condition the output of a hole-filling generator. Ulyanov et al. [10] further found that the network needs no external dataset training and can rely on the structure of the generative network itself to complete the corrupted image. However, this approach can require a different set of hyper parameters for every image, and applies several iterations to achieve good results. Moreover, their design [10] is not able to use skip links, which are known to produce detailed output. With standard convolutional layers, the raw features of noise or wrong hole initialization values in the encoder stage will propagate to the decoder stage. Our work also does not depend on placeholder values in the hole regions, but we also aim to achieve good results in a single feedforward pass and enable the use of skip links to create detailed predictions.

Our work makes extensive use of a masked or reweighted convolution operation, which allows us to condition output only on valid inputs. Harley et al. [19] recently made use of this approach with a soft attention mask for semantic segmentation. It has also been used for full-image generation in PixelCNN [20], to condition the next pixel only on previously synthesized pixels. The partial convolution can be seen as a special case of the normalized convolution, as introduced in [21]. Our primary contribution with respect to the partial convolution is to further update the input mask for the next layer based on where the partial convolution was able to make a valid response, and in applying partial convolutions to the inpainting problem. Our full model is an encoder-decoder architecture with sufficient receptive fields such that the mask is fully valid before it enters the decoder half, which simplifies the decoding process.

3 Approach

Our proposed model uses stacked partial convolution operations and mask updating steps to perform image inpainting. We first define our convolution and mask update mechanism, then discuss model architecture and loss functions.

3.1 Partial Convolutional Layer

For brevity, we refer to our partial convolution operation and mask update function jointly as the *Partial Convolutional Layer*.

Let $\mathbf{W}$ be the convolution filter weights for the convolution filter and b its the corresponding bias. $\mathbf{X}$ are the feature values (pixels values) for the current convolution (sliding) window and $\mathbf{M}$ is the corresponding binary mask. The partial convolution at every location, similarly defined in [9], is expressed as:

$$x' = \begin{cases} \mathbf{W}^T(\mathbf{X} \odot \mathbf{M})\frac{1}{\text{sum}(\mathbf{M})} + b, & \text{if sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $\odot$ denotes element-wise multiplication. As can be seen, output values depend only on the unmasked inputs. The scaling factor $1/\text{sum}(\mathbf{M})$ applies appropriate scaling to adjust for the varying amount of valid (unmasked) inputs.

After each partial convolution operation, we then update our mask. Our unmasking rule is simple: if the convolution was able to condition its output on at least one valid input value, then we remove the mask for that location. This is expressed as:

$$m' = \begin{cases} 1, & \text{if sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

and can easily be implemented in any deep learning framework as part of the forward pass. With sufficient successive applications of the partial convolution layer, any mask will eventually be all ones, if the input contained any valid pixels.

3.2 Network Architecture and Implementation

Implementation. Partial convolution layer is implemented by extending existing standard PyTorch[22], although it can be improved both in time and space using custom layers. The straightforward implementation is to define binary masks of size C×H×W, the same size with their associated images/features, and then to implement mask updating is implemented using a fixed convolution layer, with the same kernel size as the partial convolution operation, but with weights identically set to 1 and bias set to 0. The entire network inference on a 512×512 image takes 0.23s on a single NVIDIA V100 GPU, regardless of the hole size.

Network Design. We design a UNet-like architecture [23] similar to the one used in [24], replacing all convolutional layers with partial convolutional layers and using nearest neighbor up-sampling in the decoding stage. ReLU is used in the encoding stage and LeakyReLU with $alpha = 0.2$ is used between all decoding layers. Batch normalization layer [25] is used between each partial convolution layer and ReLU/LeakyReLU layer except the first and last partial convolutional layers. The encoder comprises eight partial convolutional layers with stride=2. The kernel sizes are 7, 5, 5, 3, 3, 3, 3 and 3. The channel sizes are 64, 128, 256, 512, 512, 512, 512, and 512. The decoder include 8 upsampling layers, each with a factor of 2, and followed by a partial convolutional layer. The output channel for partial convolutional layers in the decoder are 512, 512, 512, 512, 256, 128, 64, and 3. The skip links feeding the decoder stage concatenate both the feature maps and binary masks channel-wise before being input to the next partial convolution layer. The last partial convolution layer's input will contain the concatenation of the original input image with hole and original mask. This makes it possible for the model to simply copy non-hole pixels.

Partial Convolution as Padding. We do not use any existing padding schemes for our convolutions when near image boundaries. Instead, the partial convolution layer directly handles this with appropriate masking. This will further ensure that the inpainted content at the image border will not be affected by invalid values outside of the image, which can be interpreted as another hole.

3.3 Loss Functions

Our loss functions target both per-pixel reconstruction accuracy as well as composition, i.e. how smoothly the predicted hole values transition into their surrounding context.

Given input image with hole $\mathbf{I}_{in}$, initial binary mask $\mathbf{M}$ (0 for holes)the network prediction $\mathbf{I}_{out}$, and the ground truth image $\mathbf{I}_{gt}$, we first define our per-pixel losses $\mathcal{L}_{hole} = \|(1 - M) \odot (I_{out} - I_{gt})\|_1$ and $\mathcal{L}_{valid} = \|M \odot (I_{out} - I_{gt})\|_1$. These are the L^1 losses on the network output for the hole and the non-hole pixels respectively.

Next, we define the perceptual loss, introduced by Gatys at al. [26]:

$$\mathcal{L}_{perceptual} = \sum_{n=0}^{N-1} \|\Psi_n(\mathbf{I}_{out}) - \Psi_n(\mathbf{I}_{gt})\|_1 + \sum_{n=0}^{N-1} \|\Psi_n(\mathbf{I}_{comp}) - \Psi_n(\mathbf{I}_{gt})\|_1 \quad (3)$$

Here, $\mathbf{I}_{comp}$ is the raw output image $\mathbf{I}_{out}$, but with the non-hole pixels directly set to ground truth. The perceptual loss computes the L^1 distances between both $\mathbf{I}_{out}$ and $\mathbf{I}_{comp}$ and the ground truth, but after projecting these images into higher level feature spaces using an ImageNet-pretrained VGG-16 [27]. Ψ_n is the activation map of the nth selected layer. We use layers $pool1$, $pool2$ and $pool3$ for our loss.

We further include the style-loss term, which is similar to the perceptual loss [26], but we first perform an autocorrelation (Gram matrix) on each feature map before applying the L^1.

$$\mathcal{L}_{style_{out}} = \sum_{n=0}^{N-1} \Big|\Big| K_n \big(\big(\Psi_n(\mathbf{I}_{out})\big)^\intercal \big(\Psi_n(\mathbf{I}_{out})\big) - \big(\Psi_n(\mathbf{I}_{gt})\big)^\intercal \big(\Psi_n(\mathbf{I}_{gt})\big) \big) \Big|\Big|_1 \tag{4}$$

$$\mathcal{L}_{style_{comp}} = \sum_{n=0}^{N-1} \Big|\Big| K_n \big(\big(\Psi_n(\mathbf{I}_{comp})\big)^\intercal \big(\Psi_n(\mathbf{I}_{comp})\big) - \big(\Psi_n(\mathbf{I}_{gt})\big)^\intercal \big(\Psi_n(\mathbf{I}_{gt})\big) \big) \Big|\Big|_1 \tag{5}$$

Here, we note that the matrix operations assume that the high level features $\Psi(x)_n$ is of shape $(H_n W_n) \times C_n$, resulting in a $C_n \times C_n$ Gram matrix, and K_n is the normalization factor $1/C_n H_n K_n$ for the nth selected layer. Again, we include loss terms for both raw output and composited output.

Our final loss term is the total variation (TV) loss $\mathcal{L}_{tv}$: which is the smoothing penalty [28] on P, where P is the region of 1-pixel dilation of the hole region.

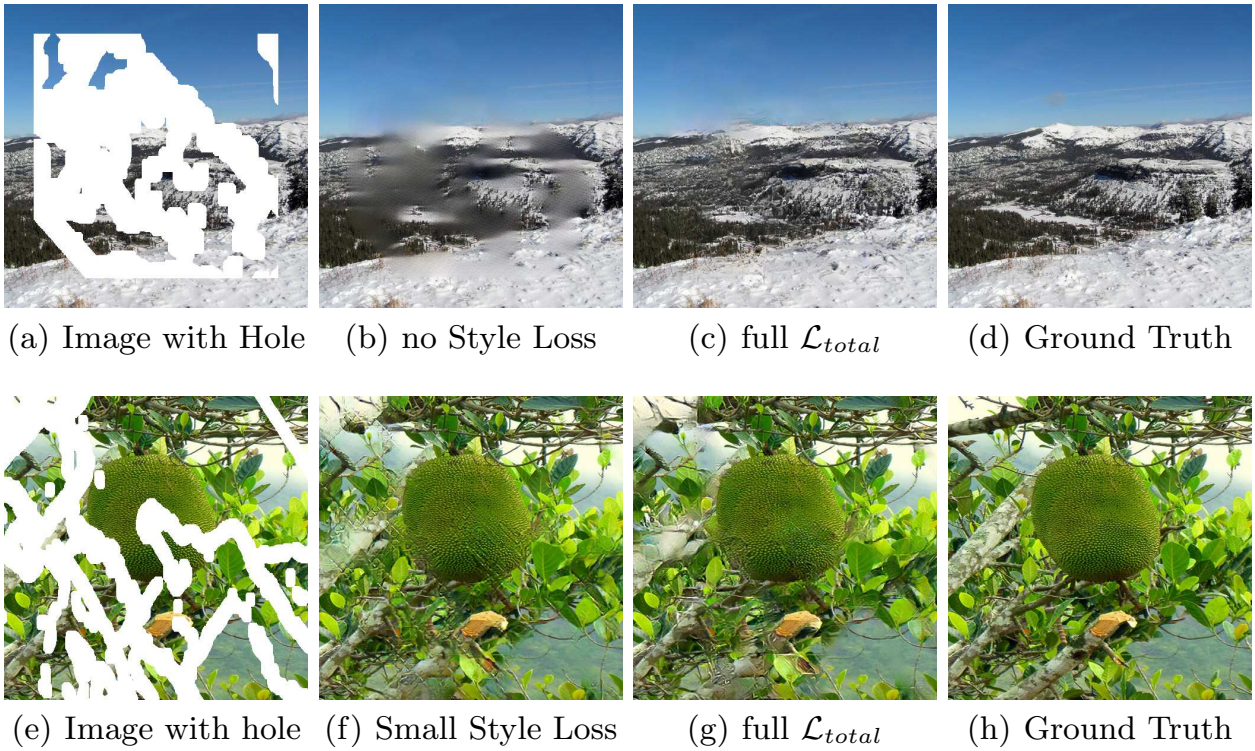
$$\mathcal{L}_{tv} = \sum_{(i,j)\in P,(i,j+1)\in P} \|\mathbf{I}^{i,j+1}_{comp} - \mathbf{I}^{i,j}_{comp}\|_1 + \sum_{(i,j)\in P,(i+1,j)\in P} \|\mathbf{I}^{i+1,j}_{comp} - \mathbf{I}^{i,j}_{comp}\|_1 \tag{6}$$

The total loss $\mathcal{L}_{total}$ is the combination of all the above loss functions.

$$\mathcal{L}_{total} = \mathcal{L}_{valid} + 6\mathcal{L}_{hole} + 0.05\mathcal{L}_{perceptual} + 120(\mathcal{L}_{style_{out}} + \mathcal{L}_{style_{comp}}) + 0.1\mathcal{L}_{tv} \tag{7}$$

The loss term weights were determined by performing a hyperparameter search on 100 validation images.

Removing Checkerboard Artifacts and Fish Scale Artifacts. Perceptual loss [28] is known to generate *checkerboard artifacts*. Johnson et al. [28] suggests to ameliorate the problem by using the total variation (TV) loss. We found this not to be the case for our model. Figure 3(b) shows the result of the model trained by removing $\mathcal{L}_{style_{out}}$ and $\mathcal{L}_{style_{comp}}$ from $\mathcal{L}_{total}$. For our model, the additional style loss term is necessary. However, not all the loss weighting schemes for the style loss will generate plausible results. Figure 3(f) shows the result of the model trained with a small style loss weight. Compared to the result of the model trained with full $\mathcal{L}_{total}$ in Figure 3(g), it has many *fish scale artifacts*, or blocky checkerboard artifacts. Ultimately, a style-loss weight too large will result in the loss of high frequency information. We hope this discussion will be useful to readers interested in employing VGG-based high level losses.



(a) Image with Hole (b) no Style Loss (c) full $\mathcal{L}_{total}$ (d) Ground Truth

(e) Image with hole (f) Small Style Loss (g) full $\mathcal{L}_{total}$ (h) Ground Truth

Fig. 3. In top row, from left to right: input image with hole; result without style loss; result with style loss; ground truth image. In bottom row, from left to right: input image with hole; result using small style loss weight; result using full $\mathcal{L}_{total}$; ground truth image.

4 Experiments

4.1 Irregular Mask Dataset

Previous works generate holes in their datasets by randomly removing rectangular regions within their image. We consider this insufficient in creating the diverse hole shapes and sizes that we need. As such, we begin by collecting masks of random streaks and holes of arbitrary shapes. We found the results of occlusion/dis-occlusion mask estimation method between two consecutive frames for videos described in [29] to be a good source of such patterns. We generate 55,116 masks for the training and 24,866 masks for testing. During training, we augment the mask dataset by randomly sampling a mask from 55,116 masks and later perform random dilation, rotation and cropping. All the masks and images for training and testing are with the size of 512×512.

We create a test set by starting with the 24,866 raw masks and adding random dilation, rotation and cropping. Many previous methods such as [1] have degraded performance at holes near the image borders. As such, we divide the test set into two: masks with and without holes close to border. The split that has holes distant from the border ensures a distance of at least 50 pixels from the border.

We also further categorize our masks by hole size. Specifically, we generate 6 categories of masks with different hole-to-image area ratios: (0.01, 0.1], (0.1, 0.2], (0.2, 0.3], (0.3, 0.4], (0.4, 0.5], (0.5, 0.6]. Each category contains 1000 masks with

and without border constraints. In total, we have created $6 * 2 * 1000 = 12,000$ masks. Some examples of each category's masks can be found in Figure 4.

Fig. 4. Some test masks for each hole-to-image area ratio category. 1, 3 and 5 are shown using their examples with border constraint; 2, 4 and 6 are shown using their examples without border constraint.

4.2 Training Process

Training Data We use 3 separate image datasets for training and testing: ImageNet dataset [16], Places2 dataset [30] and CelebA-HQ [31, 32]. We use the original train, test, and val splits for ImageNet and Places2. For CelebA-HQ, we randomly partition into 27K images for training and 3K images for testing.

Training Procedure. We initialize the weights using the initialization method described in [33] and use Adam [34] for optimization. We train on a single NVIDIA V100 GPU (16GB) with a batch size of 6.

Initial Training and Fine-Tuning. Holes present a problem for Batch Normalization because the mean and variance will be computed for hole pixels, and so it would make sense to disregard them at masked locations. However, holes are gradually filled with each application and usually completely gone by the decoder stage.

In order to use Batch Normalization in the presence of holes, we first turn on Batch Normalization for the initial training using a learning rate of 0.0002. Then, we fine-tune using a learning rate of 0.00005 and freeze the Batch Normalization parameters in the encoder part of the network. We keep Batch Normalization enabled in the decoder. This not only avoids the incorrect mean and variance issues, but also helps us to achieve faster convergence. ImageNet and Places2 models train for 10 days, whereas CelebA-HQ trains in 3 days. All fine-tuning is performed in one day.

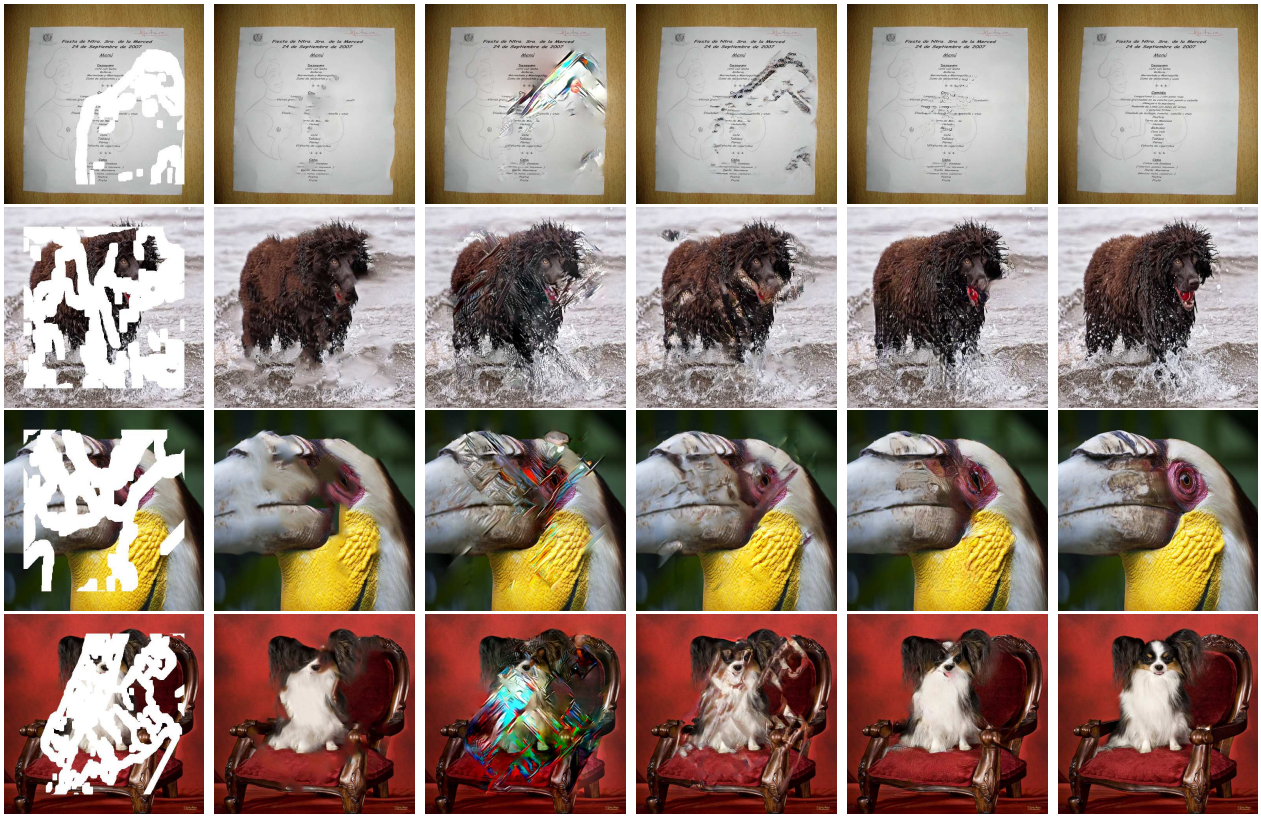
4.3 Comparisons

We compare our method with 4 methods:

- **PM**: PatchMatch [3], the state-of-the-art non-learning based approach
- **GL**: Method proposed by Iizuka et al. [1]
- **GntIpt**: Method proposed by Yu et al. [2]
- **Conv**: Same network structure as our method but using typical convolutional layers. Loss weights were re-determined via hyperparameter search.

Our method is denoted as **PConv**. A fair comparison with GL and GntIpt would require retraining their models on our data. However, the training of both approaches use local discriminators assuming availability of the local bounding boxes of the holes, which would not make sense for the shape of our masks. As such, we directly use their released pre-trained models[1]. As we do not know their train-test splits, our own splits will likely differ from theirs. We evaluate on 12,000 images randomly assigning our masks to images without replacement.

Qualitative Comparisons. Figure 5 and Figure 6 shows the comparisons on ImageNet and Places2 respectively. GT represents the ground truth. We compare with GntIpt[2] on CelebA-HQ in Figure 9. GntIpt tested CelebA-HQ on 256×256 so we downsample the images to be 256×256 before feeding into their model. It can be seen that PM may copy semantically incorrect patches to fill holes, while GL and GntIpt sometimes fail to achieve plausible results through post-processing or refinement network. Figure 7 shows the results of Conv, which are with the distinct artifacts from conditioning on hole placeholder values.



(a) Input (b) PM (c) GL (d) GntIpt (e) PConv (f) GT

Fig. 5. Comparisons of testing results on ImageNet

Quantitative comparisons. As mentioned in [2], there is no good numerical metric to evaluate image inpainting results due to the existence of many possible solutions. Nevertheless we follow the previous image inpainting works [7, 2] by

[1] `https://github.com/satoshiiizuka/siggraph2017_inpainting`, `https://github.com/JiahuiYu/generative_inpainting`

(a) Input (b) PM (c) GL (d) GntIpt (e) PConv (f) GT

Fig. 6. Comparison of testing results on Places2 images

reporting ℓ_1 error, PSNR, SSIM [35], and the inception score [36]. ℓ_1 error, PSNR and SSIM are reported on Places2, whereas the Inception score (IScore) is reported on ImageNet. Note that the released model for [1] was trained only on Places2, which we use for all evaluations. Table 1 shows the comparison results. It can be seen that our method outperforms all the other methods on these measurements on irregular masks.

User Study In addition to quantitative comparisons, we also evaluate our algorithm via a human subjective study. We perform pairwise A/B tests deployed on the Amazon Mechanical Turk (MTurk) platform. We perform two different kinds of experiments: unlimited time and limited time. We also report the cases with and without holes close to the image boundaries separately. For each situation, We randomly select 300 images for each method, where each image is compared 10 times.

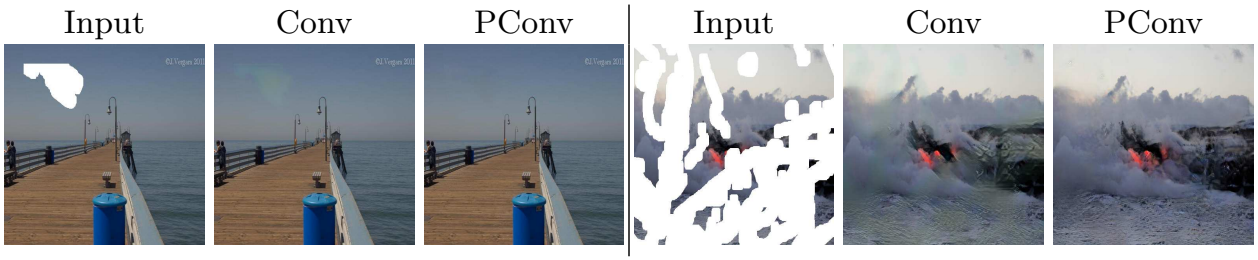


Fig. 7. Comparison between typical convolution layer based results (Conv) and partial convolution layer based results (PConv).

	[0.01,0.1]		**(0.1,0.2]**		**(0.2,0.3]**		**(0.3,0.4]**		**(0.4,0.5]**		**(0.5,0.6]**	
	N	B	N	B	N	B	N	B	N	B	N	B
ℓ_1(PM)(%)	**0.45**	**0.42**	1.25	1.16	2.28	2.07	3.52	3.17	4.77	4.27	6.98	6.34
ℓ_1(GL)(%)	1.39	1.53	3.01	3.22	4.51	5.00	6.05	6.77	7.34	8.20	8.60	9.78
ℓ_1(GnIpt)(%)	0.78	0.88	1.98	2.09	3.34	3.72	4.98	5.50	6.51	7.13	8.33	9.19
ℓ_1(Conv)(%)	0.52	0.50	1.26	1.17	2.20	2.01	3.37	3.03	4.58	4.10	6.66	6.01
ℓ_1(PConv)(%)	0.49	0.47	**1.18**	**1.09**	**2.07**	**1.88**	**3.19**	**2.84**	**4.37**	**3.85**	**6.45**	**5.72**
PSNR(PM)	32.97	33.68	26.87	27.51	23.70	24.35	21.27	22.05	19.70	20.58	17.60	18.22
PSNR(GL)	30.17	29.74	23.87	23.83	20.92	20.73	18.80	18.61	17.60	17.38	16.90	16.37
PSNR(GnIpt)	29.07	28.38	23.20	22.86	20.58	19.86	18.53	17.85	17.31	16.68	16.24	15.52
PSNR(Conv)	33.21	33.79	27.30	27.89	24.23	24.90	21.79	22.60	20.20	21.13	**18.24**	18.94
PSNR(PConv)	**33.75**	**34.34**	**27.71**	**28.32**	**24.54**	**25.25**	**22.01**	**22.89**	**20.34**	**21.38**	18.21	**19.04**
SSIM(PM)	**0.946**	**0.947**	0.861	0.865	0.763	0.768	0.666	0.675	0.568	0.579	0.459	0.472
SSIM(GL)	0.929	0.923	0.831	0.829	0.732	0.721	0.638	0.627	0.543	0.533	0.446	0.440
SSIM(GnIpt)	0.940	0.938	0.855	0.855	0.760	0.758	0.666	0.666	0.569	0.570	0.465	0.470
SSIM(Conv)	0.943	0.943	0.862	0.865	0.769	0.772	0.674	0.682	0.576	0.587	0.463	0.478
SSIM(PConv)	**0.946**	0.945	**0.867**	**0.870**	**0.775**	**0.779**	**0.681**	**0.689**	**0.583**	**0.595**	**0.468**	**0.484**
IScore(PM)	0.090	0.058	0.307	0.204	0.766	0.465	1.551	0.921	2.724	1.422	4.075	2.226
IScore(GL)	0.183	0.112	0.619	0.464	1.607	1.046	2.774	1.941	3.920	2.825	4.877	3.362
IScore(GnIpt)	0.127	0.088	0.396	0.307	0.978	0.621	1.757	1.126	2.759	1.801	3.967	2.525
IScore(Conv)	0.068	0.041	0.228	0.149	0.603	0.366	1.264	0.731	2.368	1.189	4.162	2.224
IScore(PConv)	**0.051**	**0.032**	**0.163**	**0.109**	**0.446**	**0.270**	**0.954**	**0.565**	**1.881**	**0.838**	**3.603**	**1.588**

Table 1. Comparisons with various methods. Columns represent different hole-to-image area ratios. N=no border, B=border

For the unlimited time setting, the workers are given two images at once: each generated by a different method. The workers are then given unlimited time to select which image looks more realistic. We also shuffle the image order to ensure unbiased comparisons. The results across all different hole-to-image area ratios are summarized in Fig. 8(a). The first row shows the results where the holes are at least 50 pixels away from the image border, while the second row shows the case where the holes may be close to or touch image border. As can be seen, our method performs significantly better than all the other methods (50% means two methods perform equally well) in both cases.

For the limited time setting, we compare all methods (including ours) to the ground truth. In each comparison, the result of one method is chosen and shown to the workers along with the ground truth for a limited amount of time. The workers are then asked to select which image looks more natural. This evaluates

how quickly the difference between the images can be perceived. The comparison results for different time intervals are shown in Fig. 8(b). Again, the first row shows the case where the holes do not touch the image boundary while the second row allows that. Our method outperforms the other methods in most cases across different time periods and hole-to-image area ratios.

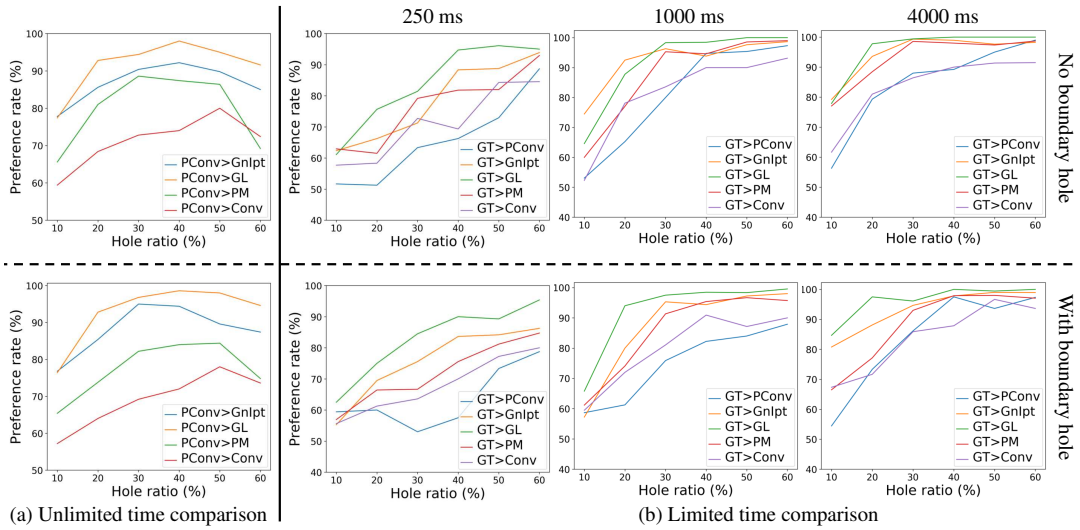


Fig. 8. User study results. We perform two kinds of experiments: unlimited time and limited time. (a) In the unlimited time setting, we compare our result with the result generated by another method. The rate where our result is preferred is graphed. 50% means two methods are equal. In the first row, the holes are not allowed to touch the image boundary, while in the second row it is allowed. (b) In the limited time setting, we compare all methods to the ground truth. The subject is given some limited time (250ms, 1000ms or 4000ms) to select which image is more realistic. The rate where ground truth is preferred over the other method is reported. The lower the curve, the better.

5 Discussion & Extension

5.1 Discussion

We propose the use of a partial convolution layer with an automatic mask updating mechanism and achieve state-of-the-art image inpainting results. Our model can robustly handle holes of any shape, size location, or distance from the image borders. Further, our performance does not deteriorate catastrophically as holes increase in size, as seen in Figure 10. However, one limitation of our method is that it fails for some sparsely structured images such as the bars on the door in Figure 11, and, like most methods, struggles on the largest of holes.

5.2 Extension to Image Super Resolution

We also extend our framework to image super resolution tasks by offsetting pixels and inserting holes. Specifically, given a low resolution image I with height H and

(a) Input (b) GntIpt (c) PConv(Ours) (d) Ground Truth

Fig. 9. Testing results on CelebA-HQ.

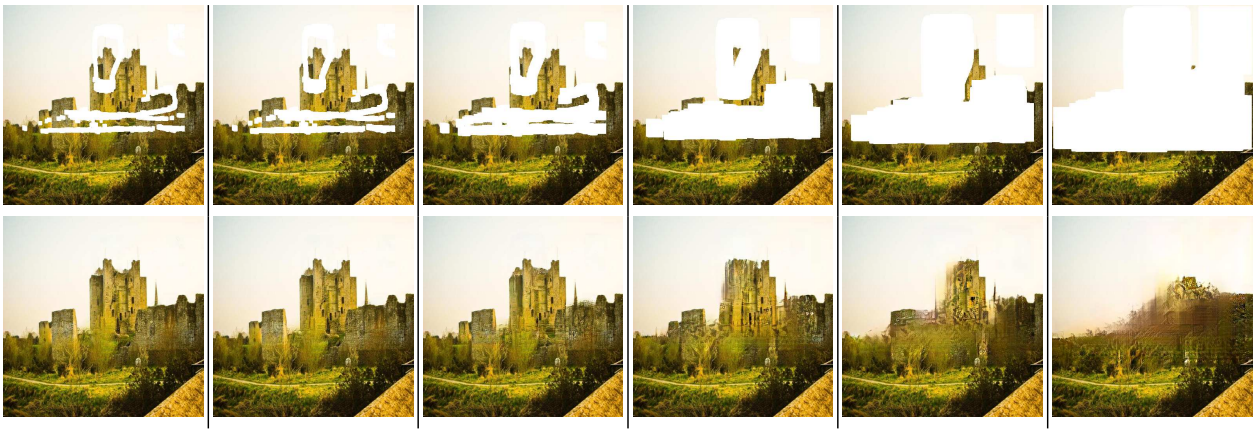
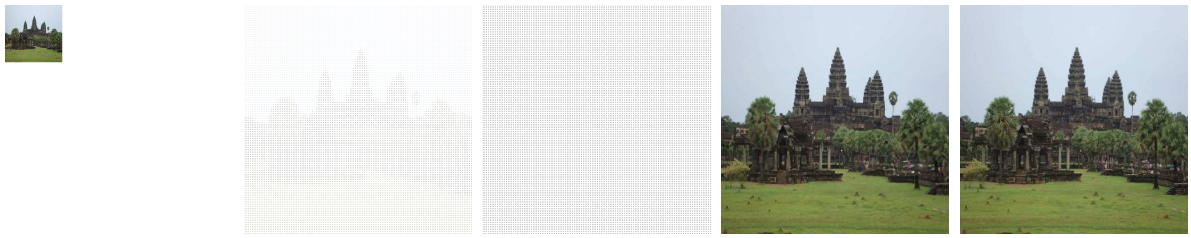


Fig. 10. Inpainting results with various dilation of the hole region from left to right: 0, 5, 15, 35, 55, and 95 pixels dilation respectively. Top row: input; bottom row: corresponding inpainted results.

width W and up-scaling factor K, we construct the input I' with height $K*H$ and width $K*W$ for the network using the following: for each pixel (x, y) in I, we put it at $(K*x+\lfloor K/2 \rfloor, K*y+\lfloor K/2 \rfloor)$ in I' and mark this position to have mask value be 1. One example input setting and corresponding output with K=4 can be found in Figure 12. We compare with two well-known image super-resolution approaches SRGAN[37] and MDSR+[38] with K=4 in Figure 13.

Fig. 11. Failure cases. Each group is ordered as input, our result and ground truth.



(a) Low Res (b) Input (c) Input Mask (d) Output (e) GT

Fig. 12. Example input and output for image super resolution task. Input to the network is constructed from the low resolution image by offsetting pixels and inserting holes using the way described in Section 5.2.

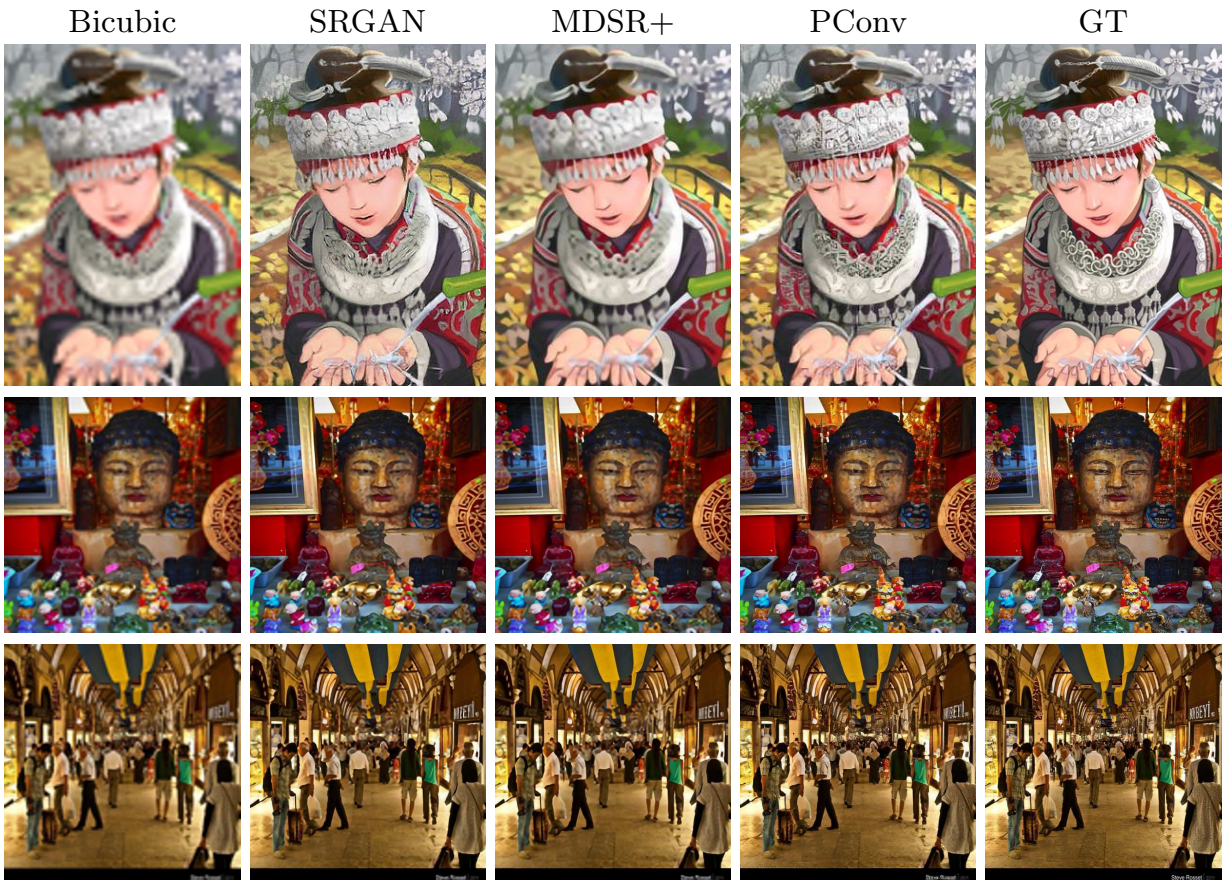


Fig. 13. Comparison with SRGAN and MDSR+ for image super resolution task.

Acknowledgement. We would like to thank Jonah Alben, Rafael Valle Costa, Karan Sapra, Chao Yang, Raul Puri, Brandon Rowlett and other NVIDIA colleagues for valuable discussions, and Chris Hebert for technical support.

References

1. Iizuka, S., Simo-Serra, E., Ishikawa, H.: Globally and locally consistent image completion. ACM Transactions on Graphics (TOG) **36**(4) (2017) 107
2. Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., Huang, T.S.: Generative image inpainting with contextual attention. arXiv preprint arXiv:1801.07892 (2018)
3. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: Patchmatch: A randomized correspondence algorithm for structural image editing. ACM Transactions on Graphics-TOG **28**(3) (2009) 24
4. Telea, A.: An image inpainting technique based on the fast marching method. Journal of graphics tools **9**(1) (2004) 23–34
5. Pérez, P., Gangnet, M., Blake, A.: Poisson image editing. ACM Transactions on graphics (TOG) **22**(3) (2003) 313–318
6. Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A.A.: Context encoders: Feature learning by inpainting. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 2536–2544
7. Yang, C., Lu, X., Lin, Z., Shechtman, E., Wang, O., Li, H.: High-resolution image inpainting using multi-scale neural patch synthesis. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Volume 1. (2017) 3
8. Hays, J., Efros, A.A.: Scene completion using millions of photographs. In: ACM Transactions on Graphics (TOG). Volume 26., ACM (2007) 4
9. Harley, A.W., Derpanis, K.G., Kokkinos, I.: Segmentation-aware convolutional networks using local attention masks. 2017 IEEE International Conference on Computer Vision (ICCV) (2017) 5048–5057
10. Ulyanov, D., Vedaldi, A., Lempitsky, V.: Deep image prior. arXiv preprint arXiv:1711.10925 (2017)
11. Bertalmio, M., Sapiro, G., Caselles, V., Ballester, C.: Image inpainting. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co. (2000) 417–424
12. Ballester, C., Bertalmio, M., Caselles, V., Sapiro, G., Verdera, J.: Filling-in by joint interpolation of vector fields and gray levels. IEEE transactions on image processing **10**(8) (2001) 1200–1211
13. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM (2001) 341–346
14. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. In: ACM Transactions on Graphics (ToG). Volume 24., ACM (2005) 795–802
15. Simakov, D., Caspi, Y., Shechtman, E., Irani, M.: Summarizing visual data using bidirectional similarity. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, IEEE (2008) 1–8
16. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3) (2015) 211–252
17. Song, Y., Yang, C., Lin, Z., Li, H., Huang, Q., Kuo, C.C.J.: Image inpainting using multi-scale feature image translation. arXiv preprint arXiv:1711.08590 (2017)
18. Yeh, R., Chen, C., Lim, T.Y., Hasegawa-Johnson, M., Do, M.N.: Semantic image inpainting with perceptual and contextual losses. arXiv preprint arXiv:1607.07539 (2016)

19. Harley, A.W., Derpanis, K.G., Kokkinos, I.: Segmentation-aware convolutional networks using local attention masks. In: IEEE International Conference on Computer Vision (ICCV). Volume 2. (2017) 7
20. van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al.: Conditional image generation with pixelcnn decoders. In: Advances in Neural Information Processing Systems. (2016) 4790–4798
21. Knutsson, H., Westin, C.F.: Normalized and differential convolution. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. (Jun 1993) 515–523
22. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. (2017)
23. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention, Springer (2015) 234–241
24. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. arXiv preprint (2017)
25. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. (2015) 448–456
26. Gatys, L.A., Ecker, A.S., Bethge, M.: A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576 (2015)
27. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
28. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: European Conference on Computer Vision, Springer (2016) 694–711
29. Sundaram, N., Brox, T., Keutzer, K.: Dense point trajectories by gpu-accelerated large displacement optical flow. In: European conference on computer vision, Springer (2010) 438–451
30. Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., Torralba, A.: Places: A 10 million image database for scene recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (2017)
31. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of International Conference on Computer Vision (ICCV). (December 2015)
32. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)
33. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. (2015) 1026–1034
34. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
35. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing **13**(4) (2004) 600–612
36. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Advances in Neural Information Processing Systems. (2016) 2234–2242
37. Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al.: Photo-realistic single image super-resolution using a generative adversarial network. arXiv preprint (2016)

38. Lim, B., Son, S., Kim, H., Nah, S., Lee, K.M.: Enhanced deep residual networks for single image super-resolution. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. Volume 1. (2017) 3

Appendix

Details of Network Architecture

Module Name	Filter Size	# Filters/Channels	Stride/Up Factor	BatchNorm	Nonlinearity
PConv1	7×7	64	2	-	ReLU
PConv2	5×5	128	2	Y	ReLU
PConv3	5×5	256	2	Y	ReLU
PConv4	3×3	512	2	Y	ReLU
PConv5	3×3	512	2	Y	ReLU
PConv6	3×3	512	2	Y	ReLU
PConv7	3×3	512	2	Y	ReLU
PConv8	3×3	512	2	Y	ReLU
NearestUpSample1		512	2	-	-
Concat1(w/ PConv7)		512+512		-	-
PConv9	3×3	512	1	Y	LeakyReLU(0.2)
NearestUpSample2		512	2	-	-
Concat2(w/ PConv6)		512+512		-	-
PConv10	3×3	512	1	Y	LeakyReLU(0.2)
NearestUpSample3		512	2	-	-
Concat3(w/ PConv5)		512+512		-	-
PConv11	3×3	512	1	Y	LeakyReLU(0.2)
NearestUpSample4		512	2	-	-
Concat4(w/ PConv4)		512+512		-	-
PConv12	3×3	512	1	Y	LeakyReLU(0.2)
NearestUpSample5		512	2	-	-
Concat5(w/ PConv3)		512+256		-	-
PConv13	3×3	256	1	Y	LeakyReLU(0.2)
NearestUpSample6		256	2	-	-
Concat6(w/ PConv2)		256+128		-	-
PConv14	3×3	128	1	Y	LeakyReLU(0.2)
NearestUpSample7		128	2	-	-
Concat7(w/ PConv1)		128+64		-	-
PConv15	3×3	64	1	Y	LeakyReLU(0.2)
NearestUpSample8		64	2	-	-
Concat8(w/ Input)		64+3		-	-
PConv16	3×3	3	1	-	-

Table 2. PConv is defined as a partial convolutional layer with the specified filter size, stride and number of filters. PConv1-8 are in encoder stage, whereas PConv9-16 are in decoder stage. The BatchNorm column indicates whether PConv is followed by a Batch Normalization layer. The Nonlinearity column shows whether and what nonlinearity layer is used (following the BatchNorm if BatchNorm is used). Skip links are shown using Concat$*$, which concatenate the previous nearest neighbor upsampled results with the corresponding mentioned PConv# results from the encoder stage.

More Comparisons on Irregular Masks

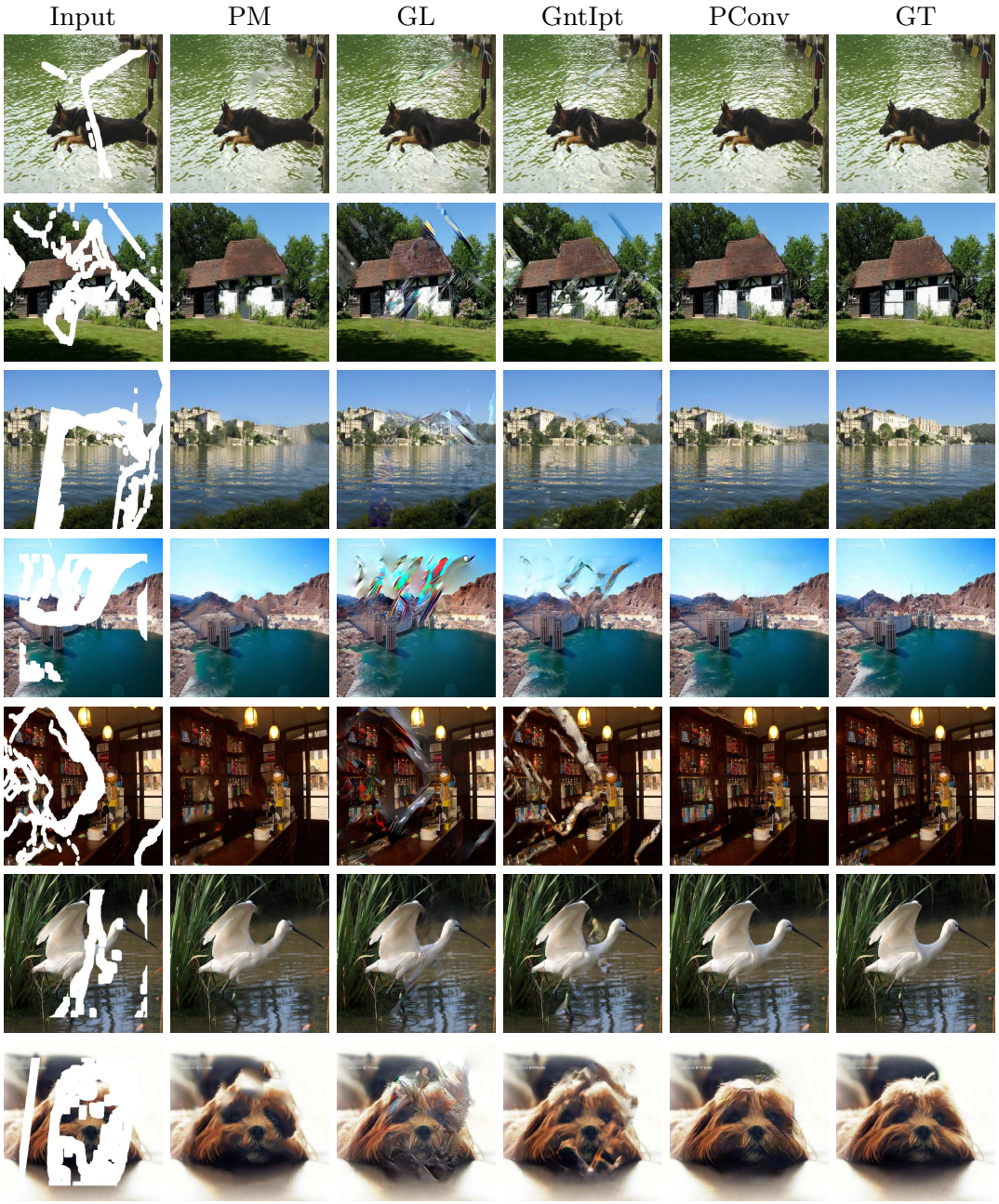


Fig. 14. Comparisons on irregular masks. The abbreviations of the notations are the same as Figure 5 and Figure 6 in the paper.

More Comparisons on Regular Masks

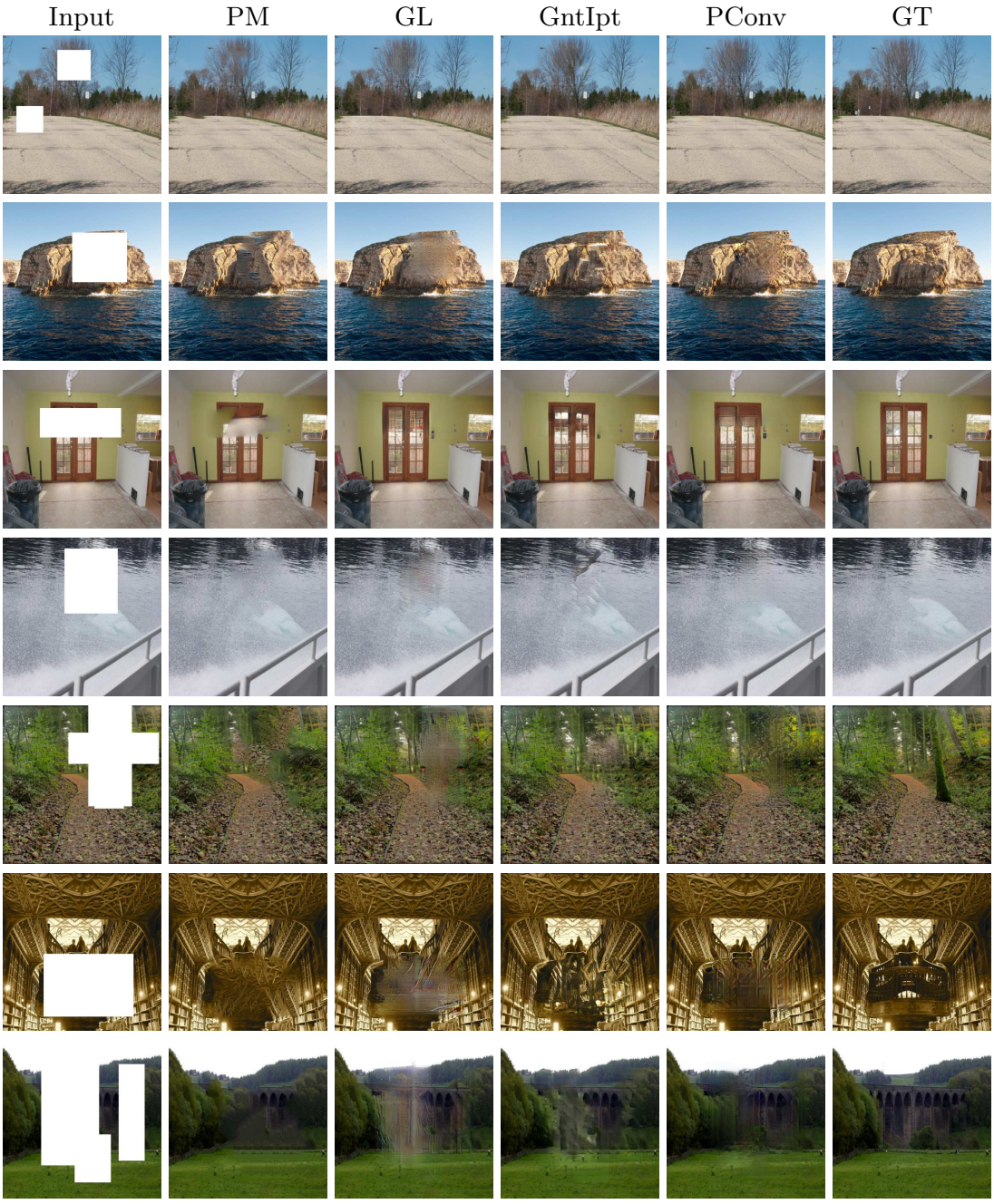
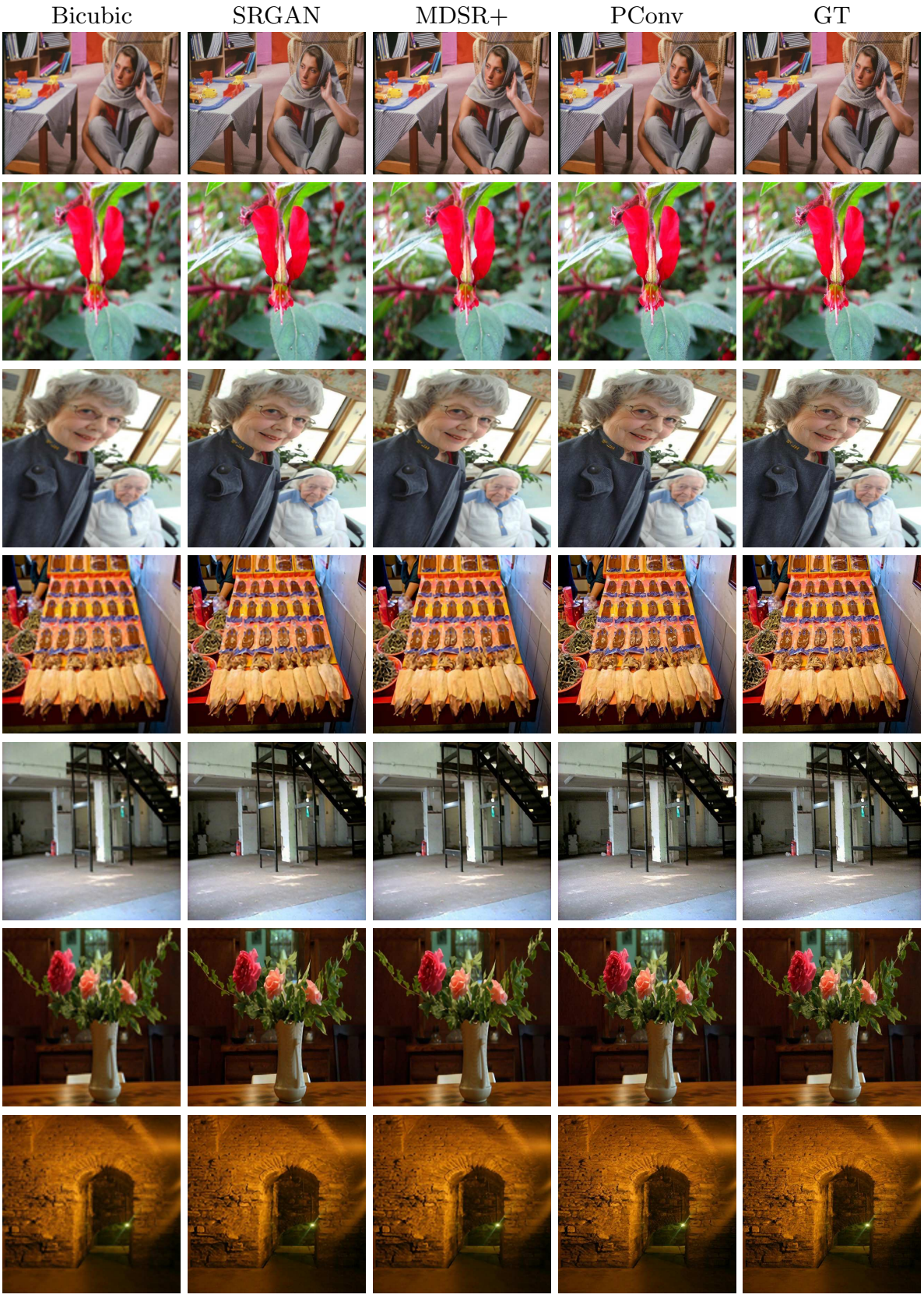
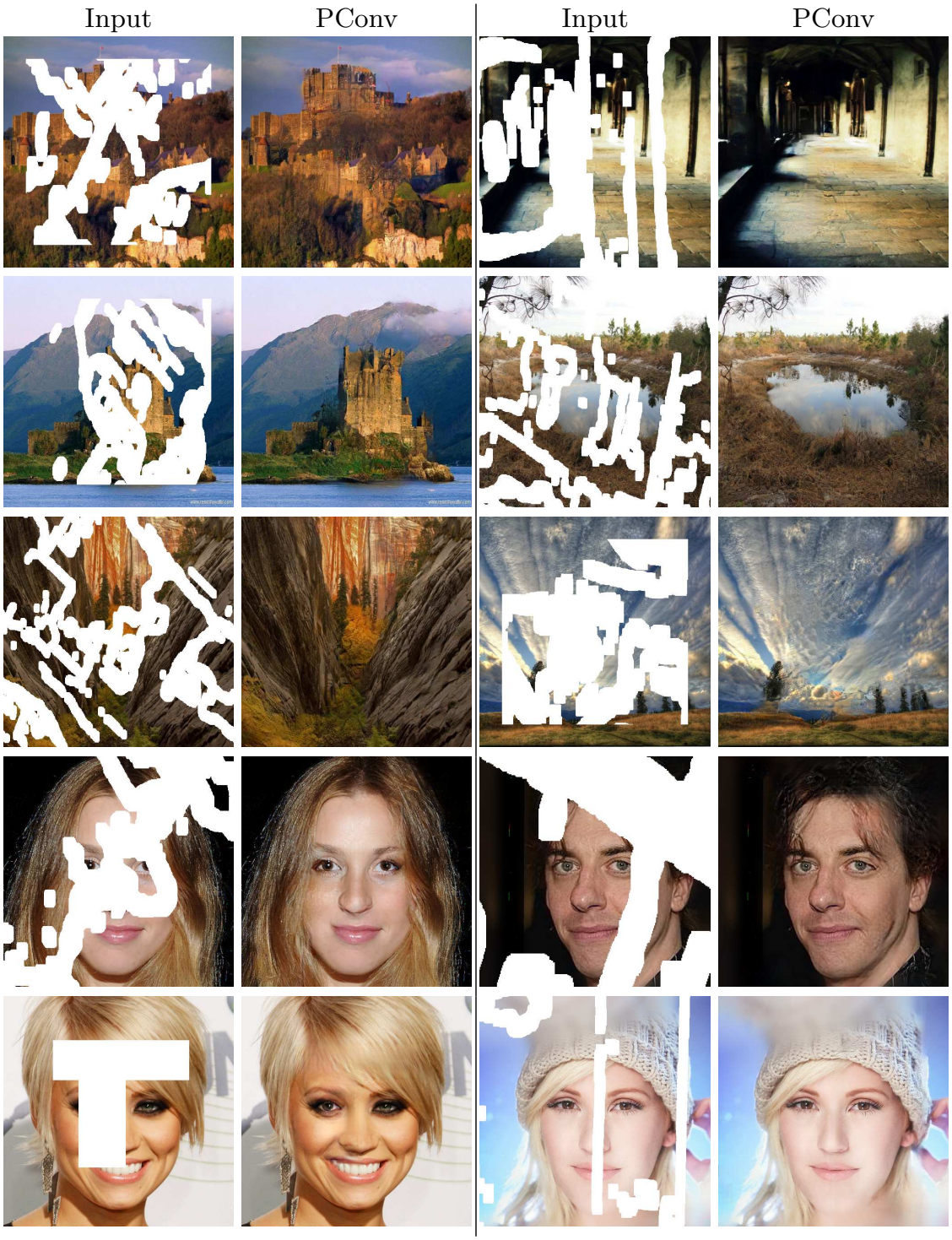


Fig. 15. Comparisons on regular masks. The abbreviations of the notations are the same as Figure 5 and Figure 6 in the paper.

More Comparisons On Image Super Resolution



More Results of Our Approach


Input
PConv
Input
PConv