Joint Gap Detection and Inpainting of Line Drawings

Kazuma Sasaki   Satoshi Iizuka   Edgar Simo-Serra   Hiroshi Ishikawa
Department of Computer Science and Engineering
Waseda University, Tokyo, Japan
milky_kaid.lc@ruri.waseda.jp   {iizuka, esimo}@aoni.waseda.jp   hfs@waseda.jp

Abstract

We propose a novel data-driven approach for automatically detecting and completing gaps in line drawings with a Convolutional Neural Network. In the case of existing inpainting approaches for natural images, masks indicating the missing regions are generally required as input. Here, we show that line drawings have enough structures that can be learned by the CNN to allow automatic detection and completion of the gaps without any such input. Thus, our method can find the gaps in line drawings and complete them without user interaction. Furthermore, the completion realistically conserves thickness and curvature of the line segments. All the necessary heuristics for such realistic line completion are learned naturally from a dataset of line drawings, where various patterns of line completion are generated on the fly as training pairs to improve the model generalization. We evaluate our method qualitatively on a diverse set of challenging line drawings and also provide quantitative results with a user study, where it significantly outperforms the state of the art.

1. Introduction

Line drawings, either drawn by pencil on paper or by digital pen on tablet, are used in many situations such as drafting or illustrating to express object contours or shapes. However, a scanned image of line drawings on paper or roughly-drawn illustrations may be missing some parts of the lines. Completing these line segments contributes in enhancing line drawings visually and in making them more precise. Line drawing is also the first step in creating color illustrations or animations, where appropriate completion of line segments are important, lest the gaps cause unintentional colorization in a later step when the flood fill (i.e., paint bucket) tool is used. However, it is onerous and time-consuming to find the gaps and manually complete the line segments while keeping natural thickness and curvature.

For natural images, various approaches to image inpainting have been proposed. For instance, methods that synthesize nearest neighbor patches [1, 17, 19] have shown especially good results in completion of natural images. Darabi et al. [5] proposed an method that considers not only the patch-color information but also the gradient of images, which conserved inpainting results better than previous works. However, these methods do not work very well when there are complex structures such as lines and curves in the removed region. As a result, they are incapable of completing line drawings adequately (Fig. 1(b)), especially in continuing the thickness and the curvature of the line. Moreover, these approaches require the missing regions specified as input, which is a real drawback in applications where there are numerous such regions. Our approach can overcome these limitations and detect and complete the gaps without any input mask specifying inpainting regions.

In this paper, we propose a novel approach for automatically filling-in the gaps in line drawings. To our knowledge, there is no known method that can naturally "inpaint" line drawings in this way. Moreover, in contrast with most inpainting methods, our approach can both automatically detect the missing line segments and complete them, adaptively recognizing and generating lines, curves, and corners suitable for the structure of the line drawing. Our approach to this problem is training a Convolutional Neural Network end-to-end with many different image patterns in a line drawing dataset. By generating training data pairs that contain many different image patterns in real-time, we train our model efficiently for the task of joint gap detection and completion.

We compare and evaluate our approach to the state of the art in inpainting, both qualitatively and quantitatively, with a user study. Our approach is preferred to others over 90% of the time, and obtains a 1.5 point advantage on a scale of 1 to 5 when absolutely scored. Note that, while our approach does not require a mask indicating what regions to inpaint, other approaches require such a mask.

In summary, the key features of our contribution are:

- a novel method for inpainting raster images of line drawings, with
- automatic detection of missing regions,
- natural continuation of thickness and curvature, and
- training data generated on-the-fly using a small dataset.

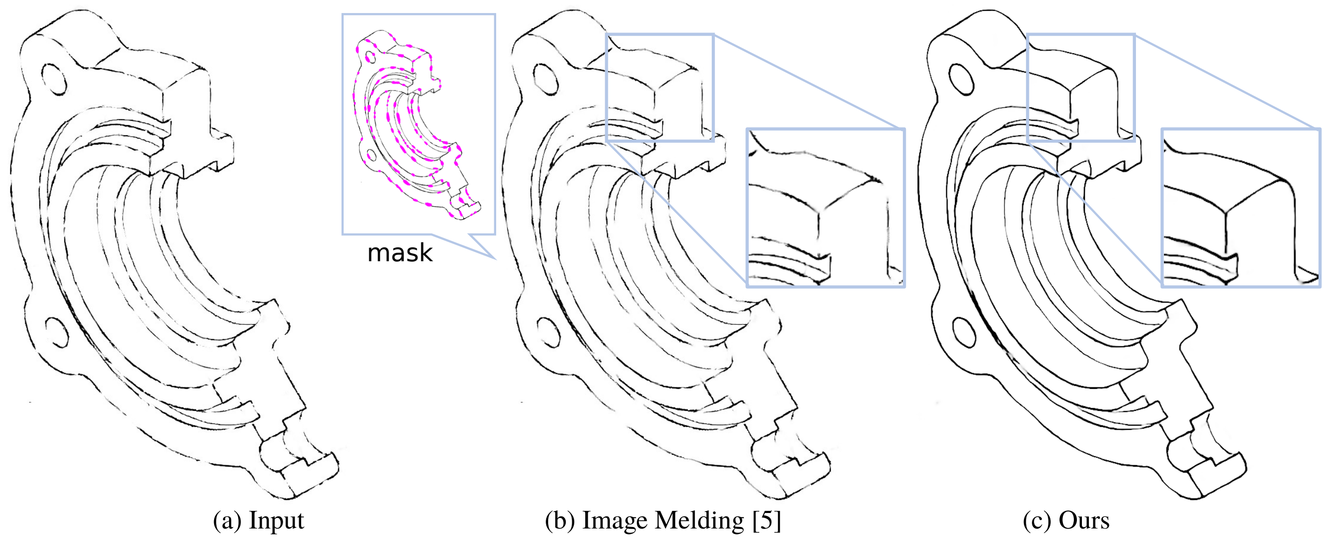


Figure 1. An example of completion of gaps in a line drawing. (a) The input raster image of a line drawing containing numerous gaps. (b) The result of inpainting with Image Melding [5], which does not generate accurate lines, even though it requires a region mask specifying where the lines are missing as an additional input (shown as the magenta mask). (c) The result by our approach, where the structure in the lines are conserved, even though it does not require the missing-region mask. Our method can correctly detect and complete the gaps in not only single lines, but also in several parallel lines and intersecting lines.

2. Related Work

One of the traditional approaches for inpainting and line-completion consists of exploiting the image curvature. Inpainting without texture [3] or using Euler's elastica [16] give good completion results for grayscale images. In a similar problem, Esedoglu et al. [7] use a Mumford-Shah model [12] for inpainting. These approaches have shown that they are capable of line completion to some extent; however, they require the missing region as input masks. Shoenemann et al. [15] propose a region-segmentation approach with curvature regularity and extend it to solve the inpainting problems, although regions that contain complex line structures are not correctly segmented. Huang et al. [8] propose a curve-completion technique for reconstructing from several image fragments. This approach needs to join the curves extracted from the image fragments one-by-one: it cannot complete several line segments contained in an image. There are other methods that take a vector image as input and complete curves by fitting with Bézier or spline curves. However, regions with intersecting lines are not completed very well, and that they require vector input falls short of the desired capability.

For inpainting more general images, approaches that utilize similar regions in an image have shown good results. In such approaches, target regions in a given image are filled using information from surrounding regions in the same image. Typically, the target regions are given as input masks, specifying gaps in old images or objects to be erased [2]. Criminisi et al. [4] presented a technique that can complete larger regions by using both texture and object structure such as the boundary. Drori et al. [6] used the level-set method to gradually complete large regions from the outside to the inside. Patch-based inpainting method [1, 5, 10, 17, 19] have shown good results for natural image completion. These methods look for patches in the same image that are most similar to the region surrounding the target, and synthesize a texture for filling-in from them. Recently, a deep learning based approach [14] has also been proposed for inpainting. However, this approach is limited to fixed size input images and focuses on inpainting a fixed sized region in the center of the image. In general, the patch-based completion approaches have difficulty completing line drawings that contain complex structures, e.g., intersecting lines and curves in the target region, as line-drawing images are dominated by white region, making it hard to detect similar patches. Additionally, they generally require the target region as the input, either given as a mask or by some user interaction. In this paper, we propose a novel data-driven approach to both automatically detect and fill the missing region of line drawings. By training a CNN with many different image patterns, we show that we can correctly complete line segments conserving thickness and curvature.

Related to this approach, we have applied deep learning to the sketch simplification problem [18]. Though we also use a fully convolutional network here, the networks differ in several aspects. At a low level, we employ upsampling layers in place of deconvolution layers, and reduce the number of filters in posterior layers, which behave in a way closer to sharpening filters than feature extractors. This results in both higher quality results and a much faster runtime. At a

Table 1. Architecture of our model. We use two different types of layers denoted as "C" for convolutional layers and "U" for nearest neighbor upsampling layers. All convolutional layers use Rectified Linear Unit (ReLU) except for the last one, for which the Sigmoid function is used to keep the output range [0,1]. Scale refers to the resolution of the output of the layer with respect to the original input size.

layer id	1	2	3	4	5	6	7	8U	8	9	10U	10	11	12U	12	13	14U	14	15	16
layer type	C	C	C	C	C	C	C	U	C	C	U	C	C	U	C	C	U	C	C	C
# filters	24	64	128	256	512	512	256	256	128	64	64	32	16	16	8	4	4	2	1	1
stride	2	2	1	2	1	2	1	-	1	1	-	1	1	-	1	1	-	1	1	1
scale	1/2	1/4	1/4	1/8	1/8	1/16	1/16	1/8	1/8	1/8	1/4	1/4	1/4	1/2	1/2	1/2	1	1	1	1

conceptual level, while both networks output line drawings, [18] is trained to simplify rough sketches using a small manually created supervised dataset, while our model is trained to both detect and inpaint line drawings, using our data generation.

3. Line Drawing Completion Method

For line-drawing inpainting, we use a deep Convolutional Neural Network (CNN). Unlike the more familiar CNN models for classification, our model produces an image output. It consists only of convolution and upsampling layers. Since it does not have fully-connected layers, the size of the input and output images are not fixed, outputting images of the same size as the input. The input is a grayscale line-drawing image, and the output is the image in which gaps in the line drawing are completed. In our approach, we can train the very deep model from scratch using a small dataset of clean line drawings by augmenting it by generating different kinds of line completion patterns automatically. We erase parts of each image and use it as the input for training, while the complete image is used as the target. By training with these pairs of target and input images with discontinuous regions, the network can learn where and what kind of lines it should complete. The real-time generation of training data by removing random parts of the images is critical in order to train our model using only a small dataset.

3.1. Model Architecture

Our model is based on the fully convolutional networks, which have been proposed for semantic segmentation [11, 13]. Since they do not have fully-connected layers, the whole network functions as a kind of filter, and can be applied to images of any size.

The structure of our model is summarized in Table 1. This model consists only of convolution layers, using neither pooling layer nor fully-connected layer. All the convolution layers use a kernel size of 3×3 except for the first layer, which uses a kernel size of 5×5. The proposed model uses zero-padding for feature maps in all convolution layers, to ensure that the image does not shrink. For example, if the size of the filter kernel is 3×3 and the stride is 1 pixel, we add a 1-pixel "margin" with value 0 to the border input maps. We use filters with a 2-pixel stride to downsample to 1/2 size in each axis, without the need for explicit pooling layers. Downsampling allows the model to compute features and recognize structures from a larger region. We additionally use the nearest neighbor upsampling approach to increase the resolution of the output. We found this upsampling to give better results than the "deconvolution" layers usually used in fully convolutional networks. The size of the output is kept the same as the size of the input by using the same number of downsampling and upsampling layers.

The general architecture follows an encoder-decoder design, in which the first half of the model serves to extract features and compress the data by reducing the resolution of the image. This allows each output pixel be computed from a larger area of the input image. The second half of the model serves to consolidate the lines and sharpen the output, which is why it has a fewer filters in comparison with the first half of the model.

For activation functions, we use the Rectified Linear Unit (ReLU) for all convolution layers, except for the last one, where the Sigmoid function is used instead, so that the output falls within a grayscale range of [0, 1]. For training, we insert a batch-normalization layer [9] after each convolution layer except for the last. These batch-normalization layers keep the mean of the output near 0 and the standard deviation near 1, which enables the network to learn effectively from scratch. Additionally, the input data is normalized by subtracting and dividing by the mean and the standard deviation of the training data, respectively.

3.2. Dataset and Training

In the training process, we use clean line-drawing images as targets and the same images with lines at some regions erased as inputs. In deep CNN models, typically millions of training data is needed for learning well from scratch. However, since there is no such dataset for line drawing completion, we generate training pairs with various completion patterns automatically from a small dataset of line drawings.

Generating Training Data. Our dataset is based on only 60 images of simple line drawings. Each image has many

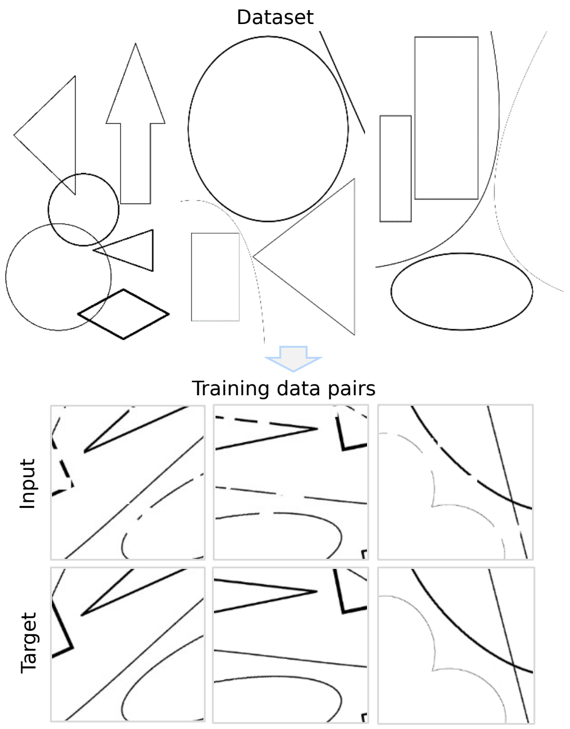


Figure 2. The process of generating training data pairs of (input, target) images from a small dataset. First, we prepare a small dataset of images of various line patterns with different thickness, curvature, and location. From this, we randomly choose an image and crop out a patch. Then, we rotate, reverse, or downscale the patch and set it as the target image. Finally, we remove the segments inside randomly-generated squares to create the input image.

kinds of lines, e.g., polygonal lines, curves, or straight lines, which can be intersecting and/or with various thickness and curvatures. When training, we generate various training data pairs as shown in Fig. 2. First, the target image of the pair is generated by randomly cropping an image patch from a dataset image. Then, the input data is created by erasing the inside of ten to twenty squares of randomly varying size (10 to 50 pixels in side) and position. Additionally, the images are rotated and scaled randomly, and flipped with one-half probability, so that more various line-completion patterns can be learned, in view of the CNN's non-invariance under rotation and inversion, as opposed to parallel translation. We note that we do not generate nor use masks for training.

Learning. We train our model using the mean squared error (MSE) as the model loss, which represents the difference between the input Y and the target Y^*:

$$l(Y^*, Y) = \frac{1}{|N|} \sum_{p \in N} (Y^*_p - Y_p)^2,$$

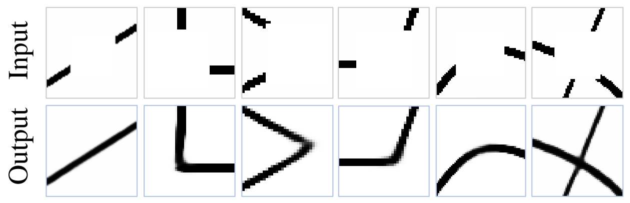


Figure 3. Examples of basic line completion results. Here, we show several zoomed-in results from larger images. In each pair, the input is shown on the top and the corresponding output on the bottom. Our model can complete well even at such places that are too complicated for most approaches, such as corners, curves with varying curvature, and intersections of lines with different thicknesses. Note that our model detects the missing area and fills it automatically, without any user input.

where N is the set of all pixels in the image, and Y^*_p and Y_p are the values at pixel p in the target and the input images, respectively. To learn the model weights, we use back-propagation with AdaDelta [20], which updates the weights without the need to manually set the learning rate.

4. Results and Discussion

Our model is trained using 352×352-pixel patches extracted from the training dataset, although it can be evaluated on images of any resolution. We train the model for 300,000 iterations with a batch size of 2, which takes roughly two days. We also apply a simple tone curve adjustment as post-processing with the same parameters for all images when showing results.

We evaluate our approach on real and synthetic images qualitatively, as well as quantitatively with a user study.

4.1. Qualitative Results

We show completion results with our model on various basic patterns in Fig. 3. Our model can complete complicated areas such as corners, curves with varying curvature, and intersections of lines with different thickness, from only very simple cues. As a result, a wide variety of line drawings can be completed, such as draft design, scanned illustration, and roughly generated line-drawing, as shown in Fig. 4. Our model can both automatically detect the missing line segments and complete them with correct thickness and curvature, even in images that contain many missing parts. Furthermore, it can complete several parallel lines, crossing lines, corners, and so on.

4.2. Comparison with the State of the Art

We compare our model with previous inpainting approaches in Fig. 5. In particular, we compare against PatchMatch [1] and Image Melding [5]. Our results show the most accurate completion of line drawings. The previous

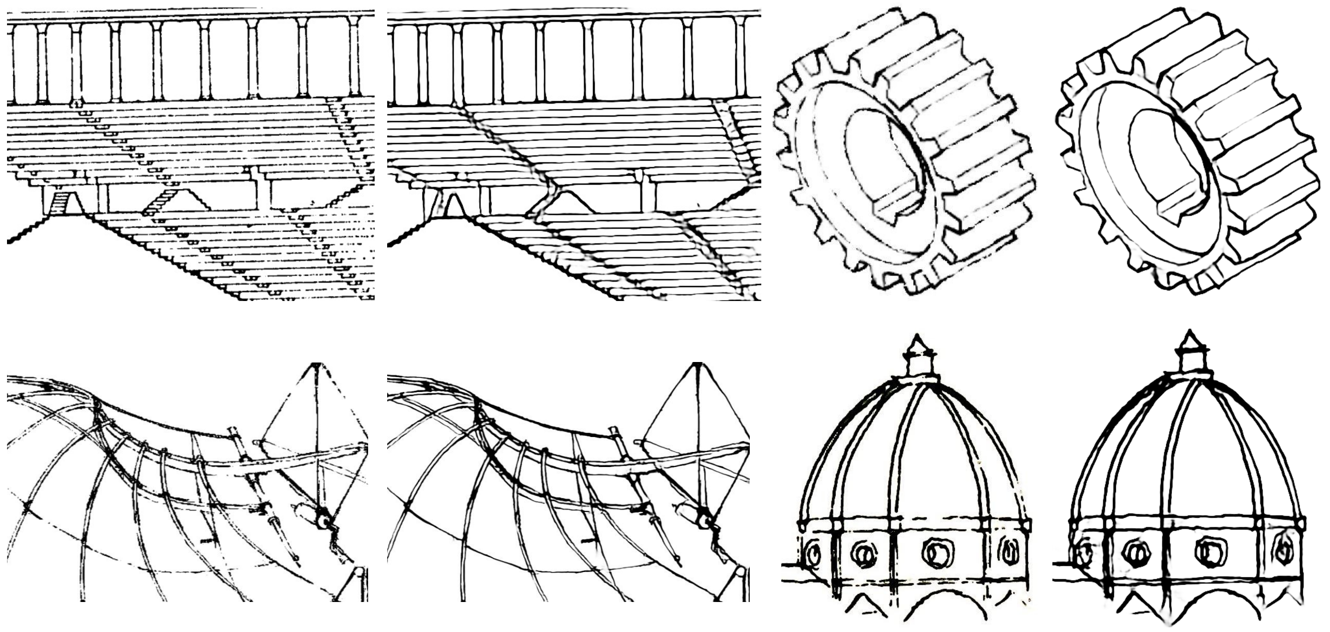


Figure 4. Results of line completion with our approach. Each image pair shows the input with missing regions on the left and the output of line completion on the right. They are all examples of real scanned data taken from various sources, including old schematics, books, and illustrations. For all the images, our proposed approach can automatically detect the missing regions and complete them, despite being from very different sources. In particular, the Leonardo da Vinci's "Flying Machine" shown in the lower left, despite having heavy deterioration, is cleanly completed by our approach.

works can complete small regions, but often fail to complete a large gap. Moreover, while these approach need the missing regions as an additional input, our method automatically detects and recognizes where the lines are missing, completing them in a fully-automatic fashion. To utilize the missing region mask with our approach, we tried restoring the image outside of the region after the completion; however, it resulted in little difference (Fig. 5 (e)).

We also compared with the deep learning based approach of Pathak et al. [14] in Fig. 6. We note that unlike our approach, it is limited to both fixed resolution input images and a fixed missing region in the center of the image, while our approach automatically detects gaps and completes them in the entire image and not limited to a predefined mask.

4.3. User Study

We also present the result of a user study using 20 images, in which we perform both relative and absolute evaluations. Half of the images are old scanned images that have deteriorated, causing gaps, while the other half consist of synthetic images with random gaps. Some of the scanned images can be seen in Fig. 4. The synthetic images are all 480×480 pixels and have 5 to 15 randomly generated square holes with 10- to 50-pixel side. For the real scanned data, we manually label input masks on all the gaps for the approaches that require them. A total of 18 users participated in the study.

Table 2. Comparison with the state of the art with relative evaluation. We show the percentage of the time the approach on the left column is preferred to the approach on the top row.

vs	[1]	[5]	Ours	Ours (Masked)
[1]	-	26.4	6.1	1.4
[5]	73.6	-	4.2	5.3
Ours	93.9	95.8	-	53.9
Ours (Masked)	98.6	94.7	46.1	-

We compare against the approaches of PatchMatch [1] and Image Melding [5] which both require the masks as user input. We additionally compare our approach when using the masks to our approach without using the masks, i.e, when it jointly detects and completes the gaps. We did not compare against Pathak et al. [14] due to its limitations in image size.

We first perform a relative evaluation in which we show users two images and ask them which one looks completed better. For each of the 20 images, we evaluate all the possible pairwise comparisons of the four approaches: PatchMatch [1], Image Melding [5], Ours, and Ours (Masked). The results are shown in Table 2. We can see that our approach is almost unilaterally chosen over the state of the art. Except for the comparison with [1], our approach performs better when not using the masks than when using them. This is an

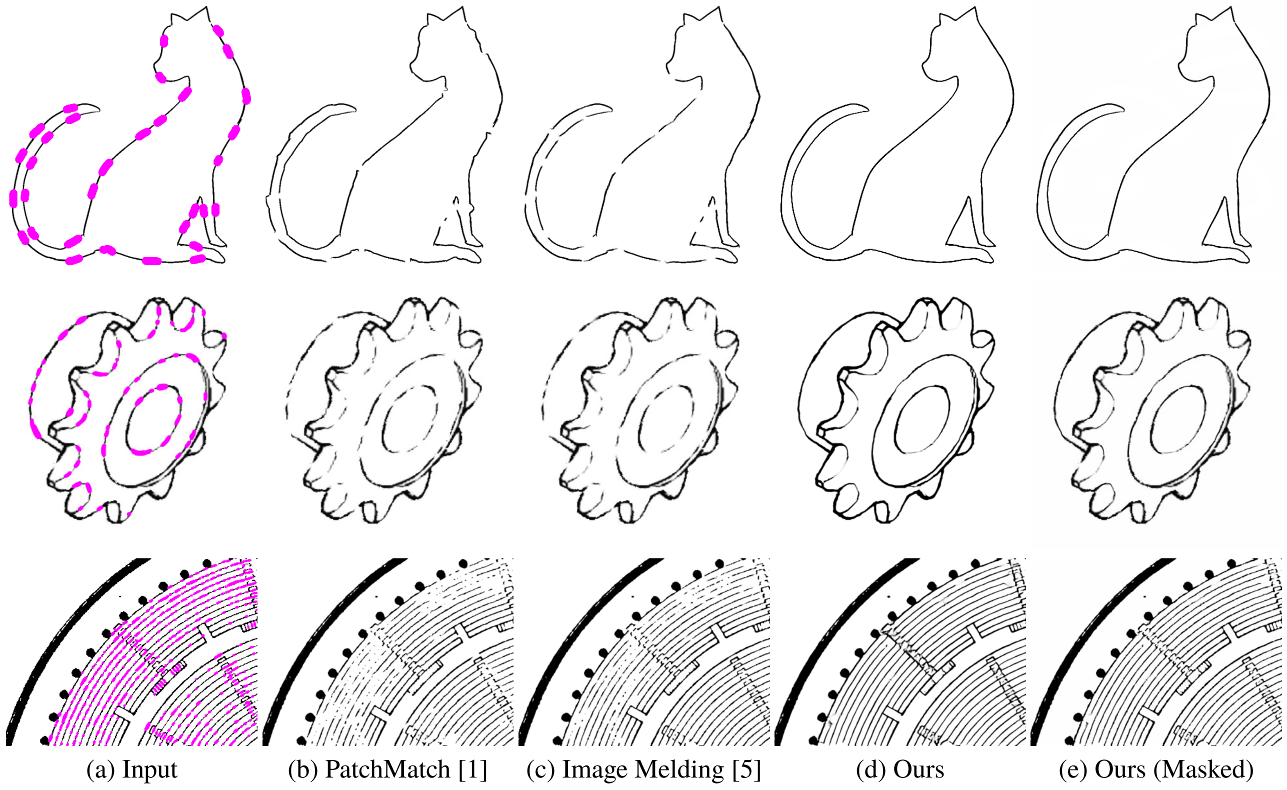


Figure 5. Comparison with the state of the art. (a) Input images with the mask overlaid in magenta. (b) Results of PatchMatch [1]. (c) Results of Image Melding [5]. (d) Our results without using the input mask, i.e., jointly detecting the gaps and completing them. (e) Our results when using the input mask. Whereas the previous works cannot correctly complete missing regions in the input image, our results show precise completion. Moreover, previous approaches require manually specifying the missing region shown as the magenta mask in (a), while our approach can both detect the gaps and complete them automatically.

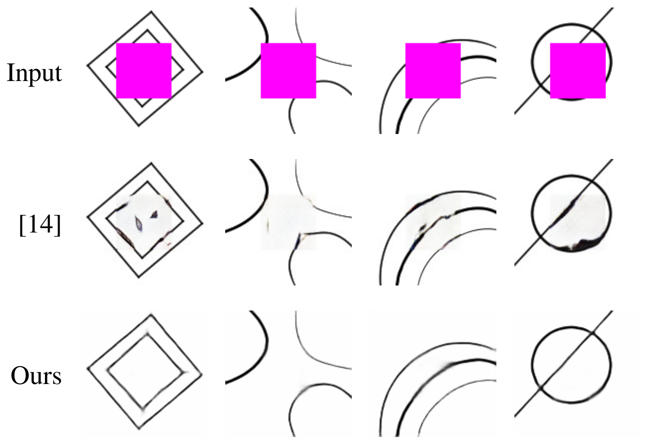


Figure 6. Comparison with the deep learning approach of Pathak et al. [14], which can only handle small fixed-sized (128 × 128 pixel) input images and only completes the fixed square at the center shown in magenta in the input images—two limitations not shared by our approach. The comparison shows that our approach can inpaint accurately without the mask as an input, while [14] adds additional elements to the image and is unable to conserve the thickness of the lines.

Table 3. Comparison with the state of the art using absolute evaluation. Each of the approaches is scored on a scale of 1 to 5. The highest values are highlighted in bold.

	[1]	[5]	Ours	Ours (Masked)
Scanned data	2.1	2.7	**3.9**	3.8
Synthetic data	2.8	2.9	**4.6**	**4.6**
Mean	2.5	2.8	**4.2**	**4.2**

interesting result that might mean our method also cleans the non-gap lines, and in strong contrast with the state of the art approaches that require the mask as input.

We additionally perform an absolute evaluation in which we show each of the users the input and output for a particular method (without the input mask) and ask them to rate the quality of line completion on a scale of 1 to 5. We show the results of this evaluation in Table 3 and Fig. 7. We can see that our approach with and without the mask obtains roughly the same score, which is significantly higher than that of the state of the art. There is also a difference between the results

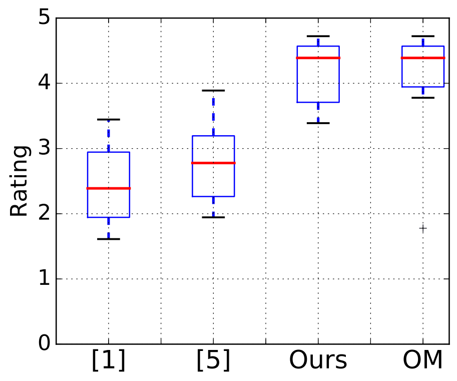


Figure 7. Box plot of the results of the absolute evaluation. Though there is no significant difference between our approach and our approach using masks (OM), our approach is significantly better than the state of the art methods PatchMatch [1] and Image Melding [5].

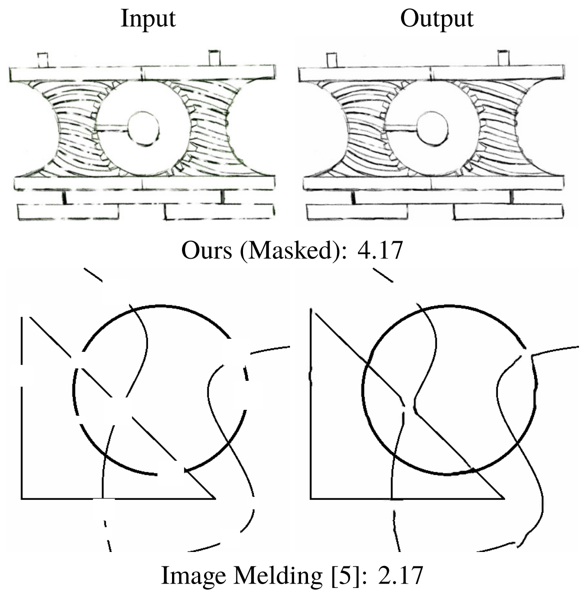


Figure 8. Two examples of images used in the absolute evaluation user study and their mean score. The top row shows a scanned image with the result and the score by our method (with mask), while the bottom row shows a synthetic image with the result and the score by Image Melding. Note that the users were not told what method was used to generate the output image.

for the 10 real scanned images and the 10 synthetic images. This may be attributed to the complexity of the real line drawings, in addition to the deteriorations from the scanning process. Examples of pairs of images shown to the users and their average scores are shown in Fig. 8.

4.4. Computation Time

We compare the running times of the existing methods in Table 4, using Intel Core i7-5960X CPU at 3.00GHz with 8 cores and NVIDIA GeForce TITAN X GPU. Unlike PatchMatch [1] and Image Melding [5], the computation time of our approach depends only on the input size, and not on the size of the area to be completed. Using the CPU, our approach takes roughly the same time as PatchMatch when completing 40% of the image. When using the GPU, our approach is 60 times faster.

Table 4. Comparison of computation time for different approaches. We evaluate for different resolution sizes and size of the input mask to be inpainted. We note that the computation time of our approach only depends on the input image resolution.

				Ours	
Size (px)	Mask	[1]	[5]	CPU	GPU
512×512	20%	0.148s	138s		
512×512	40%	0.311s	199s	0.334s	0.006s
512×512	60%	0.534s	257s		
1024×1024	20%	0.588s	535s		
1024×1024	40%	1.161s	822s	1.278s	0.021s
1024×1024	60%	1.920s	1125s		
2048×2048	20%	2.337s	2195s		
2048×2048	40%	4.547s	3403s	4.953s	0.083s
2048×2048	60%	7.698s			

4.5. Visualization and Model Optimization

In order to try to understand what the model is learning on its own, we visualize the intermediate feature layers of the model. As each layer by itself is hard to interpret, we perform PCA on all the "pixels" of each of the intermediate layers. This is in contrast to direct visualization approaches such as [21]. We then visualize the different components of the PCA to interpret results. We show the results of this process for two different images in Fig. 9 for a model with the same architecture but more filters than the one used in the rest of the results and shown in Table 1. We observe an interesting phenomena: whereas early layers contain identifiable spatial patterns in nearly all the PCA components, later layers show such patterns in only the initial PCA components.

This led us to believe that, while it is important to have the initial layers have many filters in order to detect the gaps, the model does not need many filters in the latter layers to inpaint the line segments. We used this feedback during the development of the model and could reduce the number of filters significantly, which led to a speed up of the model and a reduction of the memory used, without decreasing the performance. We do not modify the number of layers, so that each output pixel is still computed using a large region of the input image. This is particularly important for inpainting, as the area to be inpainted is highly influenced by the region around it. This leads to layers with very few filters towards

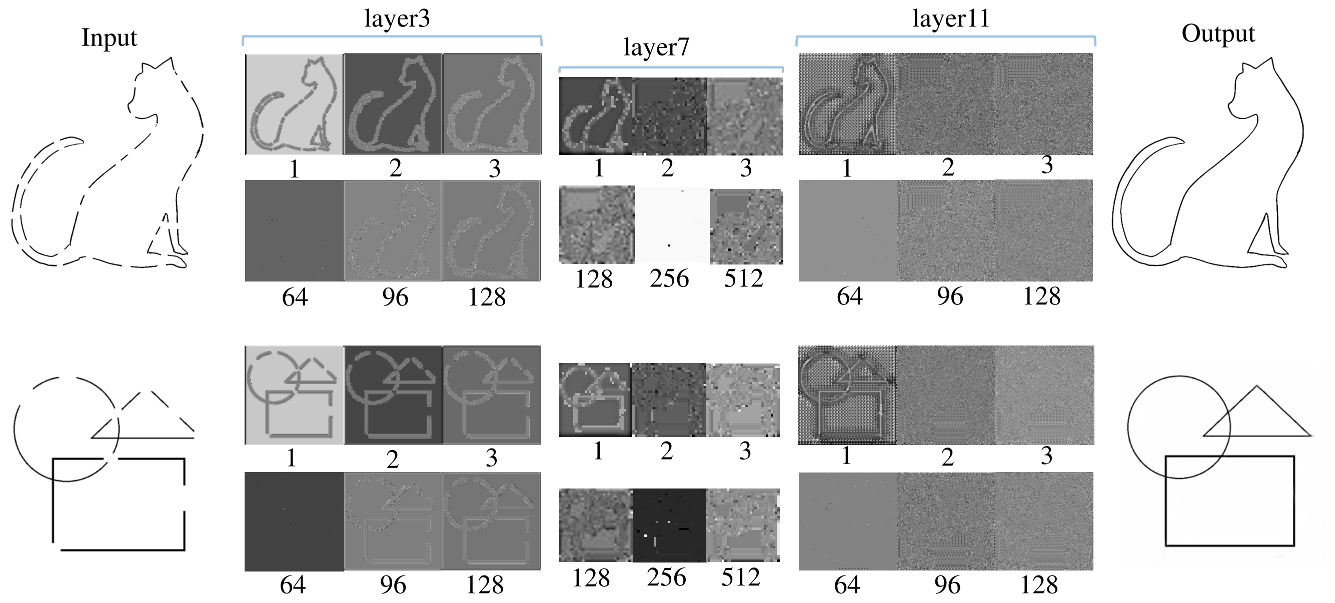
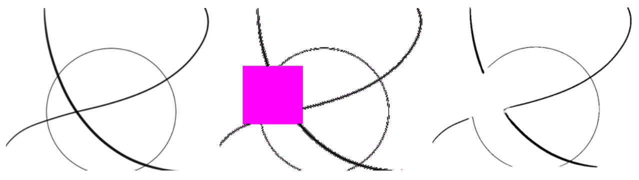


Figure 9. Visualization of the output of different internal layers of the network. We show projections onto the different principal components of the internal representations for two different images corresponding to an early layer (layer 3), a middle layer (layer 7), and a final layer (layer 11). Layers 3 and 11 are 1/4 and layer 7 is 1/16 of the size of the input image. We show the output of the convolutional layers and scale the values so they fit in the [0, 1] range for visualization purposes. We observe that while early and middle layers do conserve the structured input data in most principal components, later layers seem to concentrate all the information into the first few principal components. We use this information when deciding on the number of filters in each layer of the neural network.



(a) Ground truth (b) Input (c) Output

Figure 10. Limitations of our approach. The masked area on the input is shown as reference and is not used by our model. When complex structures with many lines and intersections must be completed, there is an inherent ambiguity to which lines must be connected. This can lead to realistic, although incorrect completions as shown in this example.

the end of the model. However, these layers can be thought as more of sharpen type filters, rather than feature extraction layers like the initial layers. Although we managed to reduce the memory usage by roughly 30% and the computation time to nearly half, it is likely further tuning of the architecture would give even better improvements. Nevertheless, the improvement in a single iteration using our visualization approach is significant.

4.6. Limitations

Though our model performs well at completing various line patterns, it can be confused by very complex structures. For example, in an image many lines intersect in a small region and that intersection is missing, it may fail to complete the gap correctly. Figure 10 shows one such example. In such a case, our model may confuse the correspondence between the line ends. Very large missing regions are also challenging because recognition of the structure is difficult. Improvement of the model loss or dataset may enable it to recognize larger regions and more complex line structures.

5. Conclusions

In this paper, we have proposed a novel data-driven approach for detecting and completing the missing regions in images of line drawings using a deep convolutional neural network. Our method automatically detects gaps and completes them with correct thickness and curvature without any user interaction. The network model is fully convolutional and therefore can handle input images of any size. We train the model efficiently with only a small dataset. Experiments and comparisons show accurate completion in various examples surpassing the previous approaches, which is corroborated by a quantitative user study in which over 90% of the users prefer our approach to previous approaches. In the future, we may improve this completion method and apply to more complex patterns by enlarging the dataset and by further improving the model architecture.

Acknowledgements This work was partially supported by JST CREST Grant No. JPMJCR14D1.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2009)*, 28(3):24:1–24:11, 2009.

[2] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 417–424, 2000.

[3] T. F. Chan and J. Shen. Nontexture inpainting by curvature-driven diffusions. *Journal of Visual Communication and Image Representation*, 12(4):436–449, 2001.

[4] A. Criminisi, P. Perez, and K. Toyama. Object removal by exemplar-based inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[5] S. Darabi, E. Shechtman, C. Barnes, D. B. Goldman, and P. Sen. Image Melding: Combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2012)*, 31(4):82:1–82:10, 2012.

[6] I. Drori, D. Cohen-Or, and H. Yeshurun. Fragment-based image completion. 22(3):303–312, 2003.

[7] S. Esedoglu and J. Shen. Digital inpainting based on the mumford–shah–euler image model. *European Journal of Applied Mathematics*, 13(04):353–370, 2002.

[8] H. Huang, K. Yin, M. Gong, D. Lischinski, D. Cohen-Or, U. M. Ascher, and B. Chen. "Mind the gap": tele-registration for structure-driven image completion. *ACM Transactions on Graphics*, 32(6):174–1, 2013.

[9] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*, 2015.

[10] N. Komodakis and G. Tziritas. Image completion using efficient belief propagation via priority scheduling and dynamic pruning. *IEEE Transactions on Image Processing*, 16(11):2649–2661, 2007.

[11] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[12] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42(5):577–685, 1989.

[13] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[14] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] T. Schoenemann, F. Kahl, and D. Cremers. Curvature regularity for region-based image segmentation and inpainting: A linear programming relaxation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2009.

[16] J. Shen, S. H. Kang, and T. F. Chan. Euler's elastica and curvature-based inpainting. *SIAM Journal on Applied Mathematics*, 63(2):564–592, 2003.

[17] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani. Summarizing visual data using bidirectional similarity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[18] E. Simo-Serra, S. Iizuka, K. Sasaki, and H. Ishikawa. Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2016)*, 35(4), 2016.

[19] Y. Wexler, E. Shechtman, and M. Irani. Space-time completion of video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):463–476, 2007.

[20] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[21] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.