Probabilistic Semantic Inpainting with Pixel Constrained CNNs

Emilien Dupont **Suhas Suresha**
Schlumberger Software Technology Innovation Center

Abstract

Semantic inpainting is the task of inferring missing pixels in an image given surrounding pixels and high level image semantics. Most semantic inpainting algorithms are deterministic: given an image with missing regions, a single inpainted image is generated. However, there are often several plausible inpaintings for a given missing region. In this paper, we propose a method to perform probabilistic semantic inpainting by building a model, based on PixelCNNs, that learns a distribution of images conditioned on a subset of visible pixels. Experiments on the MNIST and CelebA datasets show that our method produces diverse and realistic inpaintings. Further, our model also estimates the likelihood of each sample which we show correlates well with the realism of the generated inpaintings.

1 Introduction

Image inpainting algorithms find applications in many domains such as the restoration of damaged paintings and photographs (Bertalmio et al., 2000), the removal or replacement of objects in images (Liu et al., 2018) or the generation of maps from sparse measurements (Dupont et al., 2018). In these applications, a partially occluded image is passed as input to an algorithm which generates a complete image constrained by the visible pixels of the original image. As the missing or hidden regions of the image are unknown, there is an inherent uncertainty related to the inpainting of these images. For each occluded image, there are typically a large number of plausible inpaintings which both satisfy the constraints of the visible pixels and are realistic (see Fig. 1). As such, it is desirable to *sample* image inpaintings as opposed to generating them deterministically. Even though recent algorithms have shown great progress in generating realistic inpaintings, most of these algorithms are deterministic (Liu et al., 2018; Yeh et al., 2016; Yu et al., 2018a).

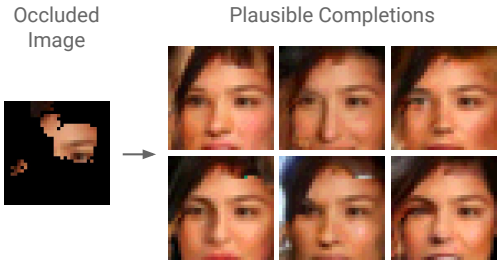


Figure 1: Example of probabilistic semantic inpainting.

In this paper, we propose a method to sample inpaintings from a distribution of images conditioned on the visible pixels. Specifically, we propose a model that simultaneously (a) generates realistic images, (b) matches pixel constraints and (c) exhibits high sample diversity. Our method, which we term Pixel Constrained CNN, is based on a modification of PixelCNNs (van den Oord et al., 2016; Oord et al., 2016) to allow for conditioning on arbitrary sets of pixels as opposed to only the ones above and to the left as in the original PixelCNN framework.

Further, our model can estimate the likelihood of generated inpaintings. This allows us, for example, to rank the various generated inpaintings by their likelihood. To the best of our knowledge, this is the first method for efficiently estimating the likelihood of inpaintings of arbitrary missing pixel regions.

To validate our method, we perform experiments on the MNIST and CelebA datasets. Our results show that the model learns to generate realistic inpaintings while exhibiting high sample diversity. Further, we also show that the likelihood estimates of the model correlate well with the realism of the generated inpaintings. Finally, we show how our model can be used to visualize individual pixel probabilities as the inpaintings are generated.

Preliminary work.

2 Related Work

Early approaches for image inpainting were mostly based on propagating the information available in the occluded image. Methods based on minimizing the total variation, for example, are able to fill small holes in an image (Shen and Chan, 2002; Afonso et al., 2011). Other methods directly propagate information from visible pixels to fill in hidden pixel regions (Bertalmio et al., 2000; Ballester et al., 2001; Telea, 2004). As these methods use only the information available in the image, they are unable to fill in large holes or holes where the color or texture have high variance. More importantly, these algorithms are also deterministic and so generate a single inpainting given an occluded image.

Other methods are based on finding patches in the occluded image or in other image datasets to infill the hidden regions (Efros and Freeman, 2001; Kwatra et al., 2005). This family of methods also includes PatchMatch (Barnes et al., 2009) which has a random component. This randomness is however limited by the possible matches that can be found in the available datasets.

Learning based approaches have also been popular for inpainting tasks (Iizuka et al., 2017; Yang et al., 2017; Song et al., 2017; Li et al., 2017; Yu et al., 2018b). Importantly, these methods often learn a prior over the image distribution and can take advantage of both this information and the pixel information available in the occluded image. Liu et al. (2018) for example achieve impressive results using partial convolutions, but these approaches are deterministic and the inpainting operation often corresponds to a forward pass of a neural network. Our method, in contrast, is able to generate several samples given a single inpainting task.

Several methods for image inpainting are also based on optimizing the latent variables of generative models. Pathak et al. (2016); Yeh et al. (2016) for example, train a Generative Adversarial Network (GAN) on unobstructed images (Goodfellow et al., 2014). Using the trained GAN, these algorithms optimize the latent variables to match the visible pixels in the occluded image. These methods are pseudo random in the sense that different initializations of the latent variable can lead to different minima of the optimization problem that matches the generated image with the visible pixels. However, the resulting completions are typically not diverse (Bellemare et al., 2017). Further, since the final images are generated by the GAN, the lack of diversity of samples sometimes observed in GANs can also be limiting (Arjovsky et al., 2017). Our approach, in contrast, is based on PixelCNNs which typically exhibit high sample diversity (Dahl et al., 2017).

3 Review of PixelCNNs

PixelCNNs (Oord et al., 2016; van den Oord et al., 2016) are probabilistic generative models which aim to learn a distribution of images $p(\mathbf{x})$. These models are based on sampling each pixel in an image conditioned on all the previously sampled pixels. Specifically, letting $\mathbf{x}$ denote the set of pixels of an n by n image and numbering the pixels from 1 to n^2 row by row (in raster scan order), we can write $p(\mathbf{x})$ as:

$$\begin{aligned} p(\mathbf{x}) &= p(x_1, x_2, ..., x_{n^2}) \\ &= \prod_{i=1}^{n^2} p(x_i | x_{i-1}, ..., x_1) \end{aligned} \tag{1}$$

We can then build a model for each pixel i, which takes as input the previous $i-1$ pixels and outputs the probability distribution $p(x_i|x_{i-1}, ..., x_1)$. We could, for example, build a CNN which takes as input the first $i-1$ pixels of an image and outputs the probability distribution over pixel intensities for the ith pixel. PixelCNNs use a hierarchy of masked convolutions to enforce this conditioning order, by masking pixels to the bottom and the right of each pixel, so that each pixel i can only access information from pixels $i-1, i-2, ...$ The model is then trained by maximizing the log likelihood on real image data.

PixelCNNs are not only used to estimate $p(\mathbf{x})$ but also to generate samples from $p(\mathbf{x})$. To generate a sample, we first initialize all pixels of an image to zero (or any other number). After a forward pass of the image through the network, the output at pixel 1 is the distribution $p(x_1)$. The value of the first pixel of the image can then be sampled from $p(x_1)$. After setting pixel 1 to the sampled value, we pass the image through the network again to sample from $p(x_2|x_1)$. We then set pixel 2 to the sampled value and repeat this procedure until all pixels have been sampled.

However, PixelCNNs can only generate images in raster scan order. For example, if pixel 3 is known, then we cannot sample $p(x_2|x_1, x_3)$ since this does not match the sampling order imposed by the masking. In image inpainting, an arbitrary set of pixels is fixed and known, so we would like to be able to sample from distributions conditioned on any subset of pixels. A trivial way to enforce this conditioning is to modify the PixelCNN architecture to take in the visible pixels as a conditioning vector (see Conditional PixelCNNs for more details (van den Oord et al., 2016)). However, our initial experiments showed that the conditioning is largely ignored and the model tends to generate images which do not match the conditioning pixels. Similar

problems have been observed when using PixelCNNs for super resolution (Dahl et al., 2017).

4 Pixel Constrained CNN

In this section we introduce Pixel Constrained CNN, a probabilistic generative model that can generate image samples conditioned on arbitrary subsets of pixels. Specifically, given a set of known constrained pixel values $\mathbf{c}$ (e.g. $\mathbf{c} = \{x_{17}, x_{52}, x_{134}\}$) we would like to model and sample from $p(\mathbf{x}|\mathbf{c})$, i.e. we would like to sample all the pixels $\mathbf{x}$ in an image, given the visible pixels $\mathbf{c}$. We factorize $p(\mathbf{x}|\mathbf{c})$ as

$$p(\mathbf{x}|\mathbf{c}) = \prod_{i:x_i \notin \mathbf{c}} p(x_i|x_{i-1}, ..., x_1, \mathbf{c}) \tag{2}$$

where the product is over all the missing pixels in the image. As noted in section 3, PixelCNNs enforce this factorization by hiding pixels with a hierarchy of masked convolutions. In the constrained pixel case, we would like to hide pixels in the same order *except* for the known pixels $\mathbf{c}$ which should be visible to all output pixel distributions. Therefore, building the constrained model amounts to using the same factorization as the original PixelCNN, but modifying the masking to make the constrained pixels visible to all pixels. This can be achieved by building a model composed of two subnetworks, a prior network and a conditioning network.

The prior network is a PixelCNN, which takes as input the full image and outputs logits which encode information from pixels $i-1, i-2, ...$ for each pixel i. The conditioning network is a CNN with regular (non masked) convolutions which takes as input the masked image, containing only the visible pixels $\mathbf{c}$, and outputs logits which encode the information in the visible pixels. Since the conditioning network does not use masked convolutions, each pixel in the logit output will have access to every visible pixel in the input (assuming the network is deep enough for the receptive field to cover the entire image).

Finally, the prior logits and conditional logits are added to output the final logits. The softmax of these logits models the probability distribution $p(x_i|x_{i-1}, ..., x_1, \mathbf{c})$ for each pixel i. This approach is illustrated in Fig. 2. Note that a similar approach has been used in the context of super resolution, where the conditioning network takes in a low resolution image instead of a masked image (Dahl et al., 2017).

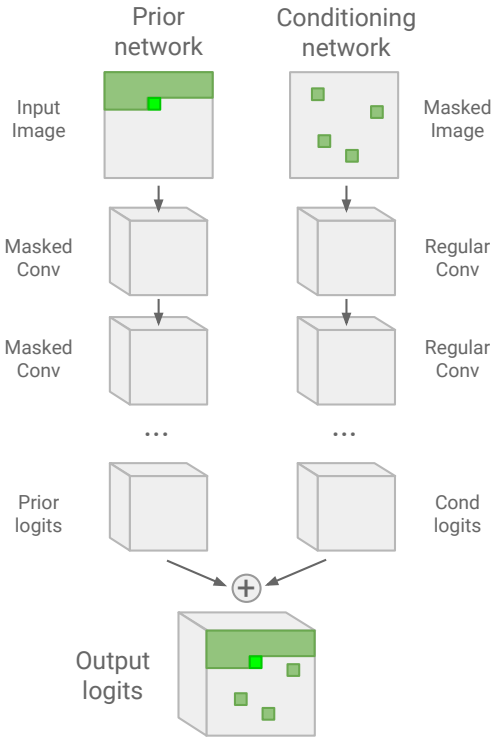


Figure 2: Pixel Constrained CNN architecture. The complete input image is passed through the prior network and the masked image is passed through the conditioning network. The light green pixel corresponds to the *ith* pixel which can access information from all the pixels in dark green, i.e $x_{i-1}, x_{i-2}, ..., x_0$ and $\mathbf{c}$.

4.1 Model Inputs

During training, the prior network takes as input a fully visible image while the conditioning network takes as input a masked version of the same image, representing the constrained pixels $\mathbf{c}$. More precisely, the constrained pixels $\mathbf{c}$ can be thought of as a function $\mathbf{c}(\mathbf{x}, M)$ of the image $\mathbf{x}$ and a mask M. The mask $M \in \{0, 1\}^{n \times n}$ is a binary matrix, where 1 represents a visible pixel and 0 a hidden pixel. The input image $\mathbf{x} \in \mathbb{R}^{n \times n \times c}$ (where c is the number of color channels) is masked by elementwise multiplication with M. To differentiate between masked pixels and pixels which are visible but have a value of 0, we append M to the masked image, so the final input to the conditioning network is in $\mathbb{R}^{n \times n \times (c+1)}$. This is illustrated in Fig. 3. The approach is similar to the one used by Zhang et al. (2016) for deep colorization.

4.2 Likelihood Maximization

We train the model by maximizing $p(\mathbf{x}|\mathbf{c})$ on a dataset of images. Ideally, the trained model should work for any set of constrained pixels $\mathbf{c}$ or, equivalently, for

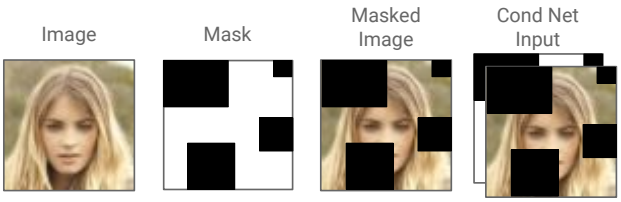


Figure 3: Conditioning network input. The input is a concatenation of the mask and the masked image in order to differentiate between hidden pixels and visible pixels which have a value of zero. This input represents the set of constrained pixels $\mathbf{c}$.

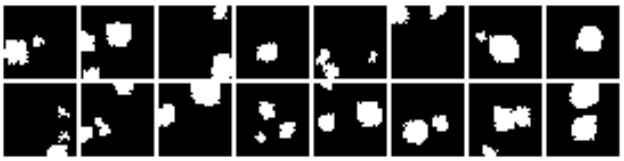


Figure 4: Examples of 32 by 32 masks generated with parameters: `max_num_blobs` $= 4$, `iter_min` $= 2$, `iter_max` $= 7$.

any mask. To achieve this, we define a distribution of masks $p(M)$ and maximize the log likelihood of our model over both the masks and the data

$$\max \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}), M\sim p(M)}[\log p(\mathbf{x}|\mathbf{c}(\mathbf{x}, M)] \tag{3}$$

When optimizing this loss in practice, we found that the conditional logits (which model the information in $\mathbf{c}$) were often partially ignored by the model. We hypothesize that this is because there is a stronger correlation between a pixel and its neighbors (which is what the prior network models) than between a pixel and the visible pixels in the image (which is what the conditioning network models). To encourage the model to use the conditional logits, we add a weighted term to the loss. Denoting by $p_{\text{cond}}(\mathbf{x}|\mathbf{c})$ the softmax of the conditional logits, the total loss is

$$\max \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}), M\sim p(M)}[\log p(\mathbf{x}|\mathbf{c}) + \alpha \log p_{\text{cond}}(\mathbf{x}|\mathbf{c})] \tag{4}$$

where α is a positive constant and the dependence of $\mathbf{c}$ on $\mathbf{x}$ and M has been omitted for clarity. This loss encourages the model to encode more information into the conditional logits and we observe that this improves performance in practice.

4.3 Random Masks

In order to evaluate the loss and train the model, we need to define the distribution of masks $p(M)$. There are several ways this can be done. For example, if it is known a priori that we are only interested in completing images which have part of their right side occluded, we can train on masks of varying width covering the right side of the image. While this is application dependent, we would like to build models that are as general as possible and can work on a wide variety of masks. Specifically, we would like our model to perform well even when missing areas are irregular and disconnected. To this end, we build an algorithm that generates irregular masks of random blobs. The algorithm randomly samples blob centers and then iteratively and randomly expands each blob. The algorithm is described in detail in Algorithm 1 and examples of the generated masks are shown in Fig. 4. While the generated masks are all irregular we find that they generalize well to completing any occlusion in unseen images, including regular occlusions.

Algorithm 1 Generate masks of random blobs.

Require: Mask height `h` and width `w`
Require: `max_num_blobs`: maximum number of blobs
Require: `iter_min`: min # of iters to expand blob
Require: `iter_max`: max # of iters to expand blob
  `mask` $\leftarrow$ zero array of size (`h`, `w`)
  `num_blobs` $\sim$ Unif(1, `max_num_blobs`)
  for $i =$ `1:num_blobs` **do**
    `num_iters` $\sim$ Unif(`iter_min`, `iter_max`)
    $x_0 \sim$ Unif(1, `w`)
    $y_0 \sim$ Unif(1, `h`)
    `mask`$[x_0, y_0] \leftarrow 1$
    `start_positions` $\leftarrow [(x_0, y_0)]$
    for $j =$ `1:num_iters` **do**
      `next_start_positions` $\leftarrow []$
      for `pos` in `start_positions` **do**
        for x, y in `neighbors(pos)` **do**
          $p \sim$ Unif$(0, 1)$
          if $p > 0.5$ **then**
            `mask`$[x, y] \leftarrow 1$
            `next_start_positions` append (x, y)
          end if
        end for
      end for
      `start_positions` $\leftarrow$ `next_start_positions`
    end for
  end for
  return `mask`

4.4 Sampling

Given a trained model and an image with a subset of visible pixels $\mathbf{c}$, we would like to generate samples from the distribution $p(\mathbf{x}|\mathbf{c})$. To generate these, we first pass the occluded image and the mask through

the conditioning network to calculate the conditional logits. When then pass a blank image through the prior network to generate the prior logits for the first pixel. If the first pixel is part of the visible pixels $\mathbf{c}$, we simply set x_1 to the value given in $\mathbf{c}$, otherwise we sample $x_1 \sim p(x_1|\mathbf{c})$ and set the value of the first pixel to x_1. We then pass the updated image through the prior network again to generate the conditional probability distribution for the second pixel and continue sampling in this way until the image is complete. Since we know the probability distribution at every pixel, we can also calculate the likelihood of the generated sample by taking the product of the probabilities of each sampled pixel.

5 Experiments

We test our model on the binarized MNIST[1] and CelebA datasets (Liu et al., 2015). As training the model is computationally intensive, we crop and resize the CelebA images to 32 by 32 pixels and quantize the colors to 5 bits (i.e. 32 colors). For both MNIST and CelebA, we use a Gated PixelCNN (van den Oord et al., 2016) for the prior network and a residual network (He et al., 2016) for the conditioning network. Full descriptions of the network architectures are given in the supplementary material.

The parameters used for generating masks are `max_num_blobs=4`, `iter_min` $= 2$, `iter_max` $= 7$. Since generating masks at every iteration is expensive, we generate a dataset of 50k masks prior to training and randomly sample these during training. The full list of training details can be found in the supplementary material.

5.1 Inpaintings

We test our models on images and masks that were not seen during training. Examples of inpaintings are shown in Fig. 5. As can be seen, the generated samples are realistic and, importantly, diverse. For example, even when the source image for the masked pixels is of a male face, the model plausibly generates a variety of both male and female face completions, each with varying hair, eye color, skin tone and so on. For MNIST, the model generates a variety of digits, all of which naturally match the conditioning.

We further test our model on rectangular masks, by occluding the top 75% of the image. Indeed, as PixelCNNs sample pixels from top to bottom, showing only the bottom pixels ensures that we are testing the conditioning network, since only it will be able to bias the top pixels based on the visible ones. Examples of inpaintings based on bottom pixels are shown in Fig. 6. As can be seen, the sampled inpaintings are plausible, match the visible pixels and are diverse. Interestingly, even though the digit which was used to generate the visible in pixels in the figure is a seven, the model is able to generate many other digit completions which plausibly match the constrained pixels. Similarly, while the faces generated in Fig. 6 share the same bottom pixels, the model generates several distinct completions.

[1]The images are binarized by setting any pixel intensity greater than 0.5 to 1 and others to 0.

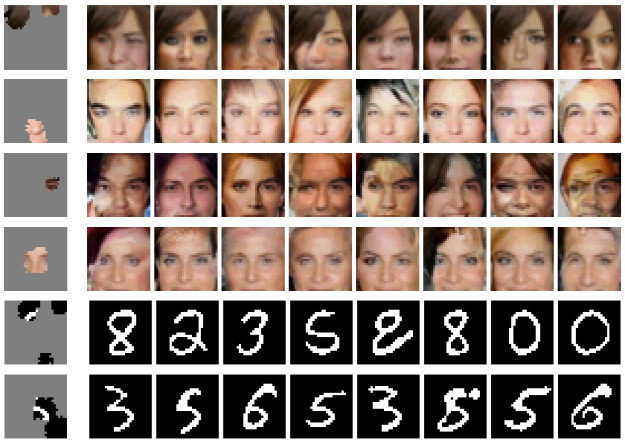
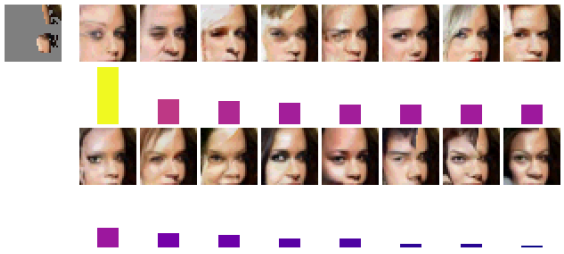


Figure 5: Inpainting results on MNIST and CelebA. The occluded images are shown on the left and various completions sampled from the model are shown on the right. As can be seen, the samples are diverse and mostly realistic.

Figure 6: Inpainting results with bottom pixels. The occluded images are shown on the left and various completions sampled from the model are shown on the right.



(a) CelebA.

(b) MNIST.

Figure 7: Inpaintings sorted by their likelihood. The occluded images are shown on the left and various completions sampled from the model are shown on the right. The size and color of the bar under each image is proportional to the likelihood of the sample. Samples with high likelihood tend to look more plausible while low likelihood samples can look unrealistic.

5.2 Inpainting Likelihood

As noted in section 4.4, our method allows us to calculate the likelihood of inpaintings. Fig. 7 shows a set of sampled inpaintings ranked by their likelihood. As can be seen, samples with high likelihood tend to look more plausible while low likelihood samples tend to look less realistic. To the best of our knowledge, this is the first method for semantic inpainting of arbitrary occlusions which also estimates the likelihood of the inpaintings. The ability to estimate inpainting likelihoods could be useful for applications where the inpainted image is used for downstream tasks which require some uncertainty quantification (Dupont et al., 2018).

5.3 Pixel Probabilities

As our model estimates the probability for each pixel $p(x_i|x_{i-1}, ..., x_1, \mathbf{c})$, we can also visualize how the pixel probabilities are affected by various occlusions. Since the MNIST images are binary, we can plot the probability of a pixel intensity being 1 for all pixels in the image, given the visible pixels. Similarly, we can observe how these probabilities change as more pixels are sampled. This is shown in Fig. 8. As can be seen, the conditional pixels bias the model towards generating digits which are plausible given the occlusion. As more

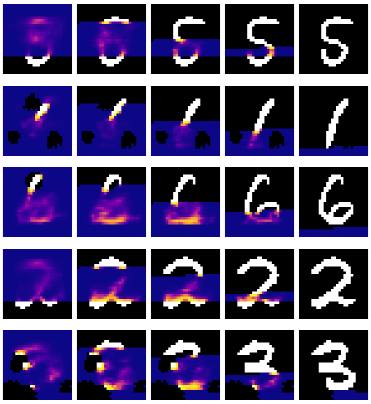


Figure 8: Pixel probability progression as pixels are sampled (figure best viewed in color). The color of each hidden pixel is proportional to the probability of that pixel being 1 (bright colors correspond to high probabilities while dark colors correspond to low probabilities).

pixels are generated, the probabilities become sharper as the model becomes more certain of which digit it will generate. For example, in the first row, the pixel probabilities suggest that both a 3, 5 or an 8 are plausible completions. As more pixels are sampled it becomes clear that a 5 is the only plausible completion and the pixel probabilities get updated accordingly.

6 Scope and Limitations

While our approach can generate a diverse set of plausible image completions and estimate their likelihood, it also comes with some drawbacks and limitations.

First, our approach is very computationally intensive both during training and sampling. As is well known, PixelCNN models tend to be very slow to train (Oord et al., 2016) which can limit the applicability of our method to large scale images. Further, most deterministic inpainting algorithms require a single forward pass of a neural net, while our model (since it is based on PixelCNNs) requires as many forward passes as

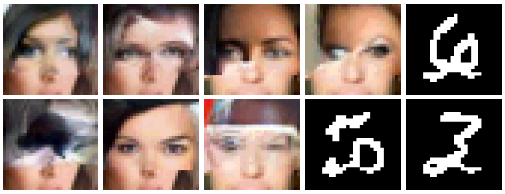


Figure 9: Failure examples. The samples are either unrealistic or do not match the constrained pixels, creating undesirable edge effects.

there are pixels in the image.

Second, our model also has failure modes where it generates implausible inpaintings or inpaintings that do not match the surrounding pixels. A few failure examples are shown in Fig. 9. These failure examples tend to have a low likelihood so this problem could be mitigated by only keeping samples which have a likelihood above a certain threshold, but they still represent a limitation of our model.

7 Conclusion

In order to address the uncertainty of image inpainting, we have introduced Pixel Constrained CNN, a model that performs probabilistic semantic inpainting by sampling images from a distribution conditioned on the visible pixels. Experiments show that our model generates plausible and diverse completions for a wide variety of regular and irregular masks on the MNIST and CelebA datasets. Further, our model also calculates the likelihood of the inpaintings which correlates well with the realism of the image completion.

In future work, it would be interesting to scale our approach to larger images by combining it with methods that aim to accelerate the training and generation of PixelCNN models (Ramachandran et al., 2017; Kolesnikov and Lampert, 2016; Reed et al., 2017). Further, it would be interesting to explore more sophisticated ways of incorporating the conditional information, such as using a weighted sum of the prior and conditional logits depending on the pixel location.

References

Afonso, M. V., Bioucas-Dias, J. M., and Figueiredo, M. A. (2011). An augmented lagrangian approach to the constrained optimization formulation of imaging inverse problems. *IEEE Transactions on Image Processing*, 20(3):681–695.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Ballester, C., Bertalmio, M., Caselles, V., Sapiro, G., and Verdera, J. (2001). Filling-in by joint interpolation of vector fields and gray levels. *IEEE transactions on image processing*, 10(8):1200–1211.

Barnes, C., Shechtman, E., Finkelstein, A., and Goldman, D. B. (2009). Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (ToG)*, 28(3):24.

Bellemare, M. G., Danihelka, I., Dabney, W., Mohamed, S., Lakshminarayanan, B., Hoyer, S., and Munos, R. (2017). The cramer distance as a solution to biased wasserstein gradients. *arXiv preprint arXiv:1705.10743*.

Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C. (2000). Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424. ACM Press/Addison-Wesley Publishing Co.

Dahl, R., Norouzi, M., and Shlens, J. (2017). Pixel recursive super resolution.

Dupont, E., Zhang, T., Tilke, P., Liang, L., and Bailey, W. (2018). Generating realistic geology conditioned on physical measurements with generative adversarial networks. *arXiv preprint arXiv:1802.03065*.

Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Iizuka, S., Simo-Serra, E., and Ishikawa, H. (2017). Globally and locally consistent image completion. *ACM Transactions on Graphics (TOG)*, 36(4):107.

Kolesnikov, A. and Lampert, C. H. (2016). Pixelcnn models with auxiliary variables for natural image modeling. *arXiv preprint arXiv:1612.08185*.

Kwatra, V., Essa, I., Bobick, A., and Kwatra, N. (2005). Texture optimization for example-based synthesis. In *ACM Transactions on Graphics (ToG)*, volume 24, pages 795–802. ACM.

Li, H., Li, G., Lin, L., and Yu, Y. (2017). Context-aware semantic inpainting. *arXiv preprint arXiv:1712.07778*.

Liu, G., Reda, F. A., Shih, K. J., Wang, T.-C., Tao, A., and Catanzaro, B. (2018). Image inpainting for irregular holes using partial convolutions. *arXiv preprint arXiv:1804.07723*.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*.

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In *Proceedings of the*

IEEE Conference on Computer Vision and Pattern Recognition, pages 2536–2544.

Ramachandran, P., Paine, T. L., Khorrami, P., Babaeizadeh, M., Chang, S., Zhang, Y., Hasegawa-Johnson, M. A., Campbell, R. H., and Huang, T. S. (2017). Fast generation for convolutional autoregressive models. *arXiv preprint arXiv:1704.06001*.

Reed, S., Oord, A. v. d., Kalchbrenner, N., Colmenarejo, S. G., Wang, Z., Belov, D., and de Freitas, N. (2017). Parallel multiscale autoregressive density estimation. *arXiv preprint arXiv:1703.03664*.

Shen, J. and Chan, T. F. (2002). Mathematical models for local nontexture inpaintings. *SIAM Journal on Applied Mathematics*, 62(3):1019–1043.

Song, Y., Yang, C., Lin, Z., Li, H., Huang, Q., and Kuo, C.-C. J. (2017). Image inpainting using multi-scale feature image translation. *arXiv preprint arXiv:1711.08590*.

Telea, A. (2004). An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34.

van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. (2016). Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798.

Yang, C., Lu, X., Lin, Z., Shechtman, E., Wang, O., and Li, H. (2017). High-resolution image inpainting using multi-scale neural patch synthesis.

Yeh, R., Chen, C., Lim, T. Y., Hasegawa-Johnson, M., and Do, M. N. (2016). Semantic image inpainting with perceptual and contextual losses. *arXiv preprint arXiv:1607.07539*.

Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., and Huang, T. S. (2018a). Free-form image inpainting with gated convolution. *arXiv preprint arXiv:1806.03589*.

Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., and Huang, T. S. (2018b). Generative image inpainting with contextual attention.

Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer.

A Model Architecture

A.1 MNIST

Prior Network

Restricted Gated Conv Block, 32 filters, 5×5
$14\times$ Gated Conv Block, 32 filters, 5×5
Conv, 2 filters, 1×1

Conditioning Network

$15\times$ Residual Blocks, 32 filters, 5×5
Conv, 2 filters, 1×1

A.2 CelebA

Prior Network

Restricted Gated Conv Block, 66 filters, 5×5
$16\times$ Gated Conv Block, 66 filters, 5×5
Conv, 1023 filters, 1×1, ReLU
Conv, 66 filters, 1×1

Conditioning Network

$17\times$ Residual Blocks, 66 filters, 5×5
Conv, 66 filters, 1×1

B Model Training

MNIST

- Optimizer: Adam
- Learning rate: 4e-4
- α: 1
- Epochs: 50

CelebA

- Optimizer: Adam
- Learning rate: 4e-4
- α: 1
- Epochs: 60

C More Samples

Figure 10: Inpaintings.

Figure 11: Likelihood sorted inpaintings.