Neural Fill:
Content Aware Image Fill with Generative Adversarial Neural Networks

Christopher Sauer, Russell Kaplan, Alexander Lin
Stanford University
450 Serra Mall, Stanford, CA 94305
{cpsauer, rjkaplan, alin719}@stanford.edu

1. Abstract

We explore the problem of content-aware image fill with convolutional neural networks. Given an image that is partially masked, our goal is to generate realistic-looking content to fill the masked parts of the image. This task is also sometimes referred to as image completion or image inpainting. We experiment with several different network architectures for the problem, and we observe our most compelling results with generative adversarial networks (GANs).

2. Introduction

Convincingly filling regions in images is a longstanding problem in computer vision. The task can be formulated as follows: given a partially masked image as input, return the image with the masked region filled in a visually plausible way.

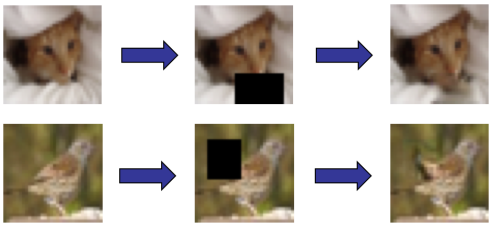

Figure 1. Example of filling a mask with plausible but different content. These examples were generated by our best two models, which created the filled image (right) from the masked image (center). For reference, the original image used is shown at left. Note that the models generated a different, but still realistic lower face for the cat, blanket folds, and bird wing position.

This problem arises in many aspects of image editing. For example, removing unwanted objects from the foreground of photos can be framed as a content-aware fill problem. So can watermark removal, restoration of damaged photographs, expansion of warped panoramas, and more.

Content-aware fill is inherently a generative problem that requires learning the structure of photographs of the real world. Given the recent successes in supervised learning—especially in comparison to unsupervised—we sought to re-frame the generative problem of content-aware fill as a supervised problem.

There is a nearly limitless supply of examples in our output space: photographs of the real world. From each of these output examples, we can generate an arbitrary number corresponding inputs by applying a random mask to the image. This procedure thus expands any dataset of images into a supervised dataset for content-aware fill, with each masked input paired with what is, by definition, a realistic completion: the original image. We describe our approach for creating these examples more in the Dataset Generation section.

3. Related Work

There is a long history of research into the image completion problem. As early as the Renaissance, human artists worked to restore gaps in damaged medieval artwork [8]. In the digital photography context, image completion is a natural desideratum of image editing software like Adobe Photoshop, where photo editors often need to crop out unwanted foreground objects while maintaining the unity of the overall image.

Previously, this problem was approached separately by two classes of algorithms: texture synthesis algorithms, which work better for repeating sample textures across large image regions, and inpainting algorithms, more suited for filling in small gaps. In texture synthesis, Ashikhmin demonstrated convincing repetitions of simple 2D patterns based on preserving L_2 similarity of pixel neighborhoods. [1] Despite convincing outputs for texture continuation, these approaches fail to convincingly fill in gaps in many real-world images, where multiple textures often interact at the gaps that need to be restored. Several algorithms

then emerged to address these cases, in particular to remove scratches and overlaid text [5] [3]. Many of these inpainting algorithms are inspired by the partial differential equations of heat flow: they attempt to propagate the multiple linear structures of the surrounding textures into the filled region via diffusion.

zMore modern approaches incorporate both texture synthesis techniques and inpainting techniques to achieve a realistic fill [7] [2]. PatchMatch, by Barnes et. al. [2], incorporated both approaches to introduce "structural image editing", a technique that allows the user to draw lines specifying the linear structures of the images to preserve, draw an image mask, and watch the algorithm fill the mask plausibly while preserving the specified linear structures. The popular Adobe Photoshop software released a "Content-Aware Fill" feature based on PatchMatch to critical acclaim in 2010.

Despite the considerable success of these modern approaches to image completion, by design they are all fundamentally constrained to repeat existing patterns and structures in an image to produce a fill. This works reasonably well for common photo editing tasks like removal of foreground objects or overlaid text, but renders these algorithms incapable of generating plausible "new" pixels in an image, based on a higher level semantic understanding of its contents. For example, given a human portrait with the head cropped out, existing techniques would not be able to fill the mask with anything resembling a human head, even though a reasoning person could infer that the crop is obstructing a head given the presence of a human torso, arms, etc. There are simply no head-like patches or patterns in the image for these existing algorithms to extend into the cropped zone.

In recent years, separate families of generative techniques have been developed by the machine learning research community that seem like a promising solution to this obstacle. Researchers have made strides recently with autoencoder models, such as the denoising autoencoder introduced by Vincent et. al. [16], which learns to reconstruct empirical data from noised inputs. Variational autoencoders, a recently introduced technique by Kingma et. al. [12], have likewise demonstrated impressive results in image generation by using neural networks to map between observed state and latent variables. It is also possible to frame the image completion problem through the lens of Recurrent Neural Networks. Oord *et al*. do so in a recent publication that frames inpainting as a pixel-by-pixel generation task using the conditional probabilities of neighboring pixels [15]. Their results are encouraging but suffer from blurriness in their output samples.

A novel approach to plausible image generation was introduced by Goodfellow et. al. [9] in 2013, known as Generative Adversarial Networks (GANs). This approach trains two networks in tandem, a generator and a discriminator. The generator is tasked with creating plausible fake images, whereas the discriminator is a binary classifier that distinguishes between real and fake images. The key architectural insight is that the generator is able to use the discriminator's gradients for learning: it updates its weights in the direction most likely to produce confusing output for discriminator. GANs have up until now been used largely for whole-image generation of new content [4]. To the best of our knowledge, we are the first to apply GANs to the problem of image completion.

4. Dataset Generation

As mentioned in the introduction, the content-aware fill problem differs from many problems in computer vision and machine learning in that the problem does not require a laboriously hand-labeled dataset. Instead, we have the luxury of being able to automatically generate inputs from outputs—photographs taken from the real world.

Given any real-world image we can generated a supervised example by randomly generating and applying a mask—we used randomly sized, randomly positioned, axis-aligned rectangles. This masked image becomes the training input, and the original image is a possible output: an image sampled from the space of real photographs with the masked region filled. There are a huge number of freely available, permissibly licensed images available on the web, and each image can be expanded into as many different inputs as required. The enormous number of training pairs, and our ability to generate them procedural means there is need to reuse the exact same example twice, and as long as we use enough source images, little risk of having the network over fit by simply memorizing the unmasked photographs.

Rather than train on a dataset of inputs and labels written to disk, we procedurally generate examples during training. This saves disk space and loading time, and vastly speeds training. Our training code for each model requests mini-batches from our ExampleGenerator. To satisfy a request for an example, the ExampleGenerator first (pseudo)randomly loads an image from the file set specified. It then draws a random mask height, H, and width, W, where $H \sim 1+B(H_i-1, 0.35)$, $W \sim 1+B(W_i-1, 0.35)$, H_i is the image height in pixels, and W_i is the image width. This yields masks that always fit in image, are non-zero, and are centered around sizes that allow enough pixels to remain so that filling is possible but difficult. The mask's position is then selected uniform from the possible locations within the image. We store this masks in a fourth color channel. Each pixel in this fourth channel is 1 if a mask is present over that pixel, and 0 otherwise. Since the whole generation process is pseudorandom with a set seed, by generating at run time we manage to reap both the benefits of example independence and run time efficiency as well as having dataset production be deterministic and reproducible.

This gives us a supervised learning pair (I_x, I_y) to return to the training model, where I_x is a four-channel masked image and I_y is the original three-channel RGB image, that is a valid fill of the original image.

Figure 2. Example of an image being randomly masked. The image on the left is the original, I_y. The center image is the four-channel, masked image, I_x. The image at the right is the color channels of the example image, visualized. Note that this example was generated with the mean-masking approach, described below.

If we simply masked the random region by setting the mask channel bits, the network could cheat by outputting the RGB channels of the input. Therefore, we need to destroy the information in the RGB channels in the masked region of the original image. Furthermore, we need to do so in a way that could be done to real images that might or might not be missing pixel values in the masked region.

Originally we did this by setting all masked pixels to the mean color of all other pixels in the original image. We compute the mean over the other pixels both to avoid leaking information about the mean color of the masked region, and to allow the operation to be applied to input images where the "masked" pixel values are genuinely unknown. However this seemed to work less well for generative adversarial networks, so we moved over to masking with unit Gaussian noise and with noise centered at the mean pixel value for those examples.

For our training we applied the ExampleGenerator to the 32x32 CIFAR-10 [13] dataset because training generative adversarial networks is extremely computationally expensive. We utilized the full 50,000 images from CIFAR, training with 85% of the images and reserving the rest for validation. However, our example generation method–and our models–could in theory be trained on images of any size. In particular, we hope to apply it to the visually-interesting MIRFLICKR-25000 dataset [11] in the future, which are perhaps more like the distribution of images you might want to apply content-aware fill to: photographs of the real world in various stages of editing. We hope to use these for generation of later stage models.

Our code for example generation is written in Lua with Torch7 [6], as is the code for our models. Since Torch reads in pixel intensities as real values on [0,1], we multiplied images by two and subtracted one to mean center them, and move the intensities to be on [-1,1]. Mean centering helps models to train, not least because padding in convolutional layers is assumed to be 0, in addition to the usual arguments about alternating gradients. We invert this transformation after filling and before visualizing images.

5. Methods, Experiments, and Results

5.1. Direct Convolution

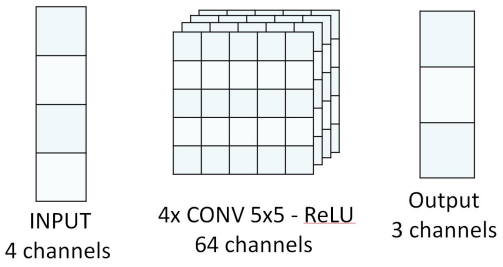

Figure 3. Highest-performing architecture for our initial Direct Convolutional network approach

As a first approach, we learned a purely convolutional (i.e. no fully-connected layers) model to minimize the standard squared loss between the generated replacements for the masked region and the masked content in the original images. The primary motivation behind this was to gain insight on how convolutional networks would behave in the generative adversarial network, and to develop the internal layers of the GAN in parallel with getting a basic GAN to train.

We began by formulating the problem as a least squares regression of the color intensities of the masked pixels. We used only convolutional layers to handle variably-sized images, and ran a variety of tests to determine the best parameters for this model. Spatial batchnorm significantly decreased performance, and that keeping a constant number of channels was the best use of parameters. Finally, using sigmoid or hard tanh layers to limit the codomain to valid images yielded little improvement. Overall, results from this approach were not impressive, and encouraged us to focus on developing the adversarial network. Some of these results are shown below.

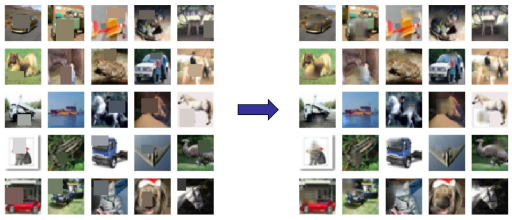

Figure 4. Images generated by our direct convolutional network

We were, however, able to determine the best architecture for this network by analyzing the losses for varying sizes (shown below). This was the most important result from these tests, and greatly informed our architecture while designing the much more successful generative adversarial network.

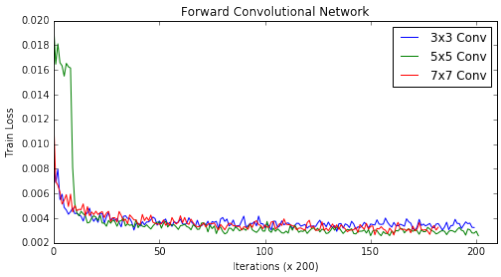

Figure 5. Loss Analysis of Various Convolutional Layer Sizes

5.2. Generative Adversarial Networks

One of the major issues encountered while using the least squares objective to generate realistic fills of image masks was that least squares often produced blurry, averaged fills of the unknown regions, rather than sharp, realistic-looking content. The reason for this is straightforward — a least squares objective penalizes outliers heavily, so the network is encouraged to smooth across many possible fills to avoid large penalties. We do not want this; our objective is to generate a fill that is realistic and plausible. We overcame this issue by using generative adversarial nets (GANs) originally proposed by Goodfellow, et al. [9].

5.2.1 The Adversarial Method

Generative adversarial nets work by training two different neural networks together: a generative net, G, whose job is to generate content, and a discriminative net, D, whose job is to distinguish the artificially-filled images of the generative net from the original, unmasked images in our dataset.

The adversarial framework for generation is also a closer fit for our original problem statement. Content-aware fill should not necessarily seek to reproduce the original, but instead re-imagine realistic content to fill the masked region. In other words, the filled region ought to be indistinguishable from a real image, given the context of the surrounding pixels. This is exactly what a generative adversarial network does: It works to find a generator whose output is indistinguishable from real images by the discriminator.

Formulating the the discriminator's objective is simple: distinguish between real and generated examples. While in principle an adversarial architecture would work with a wide variety of learners, it is usually implemented with neural networks. The discriminator is then a standard image classification network predicting a binary label, real or fake. We use a sigmoid output and the cross-entropy loss function, i.e., the output from the discriminator is the p from the sigmoid, and the loss is $-\ln(p)$ for a real example and $-\ln(1-p)$ for a fake example.

We should note that with this loss function–and no regularization–the D's output p has the nice interpretation of optimizing to the probability D would give of the image being real (given the image and the understanding of images captured in D's parameters). We can easily prove that this choice of loss incentivizes D to output such a p. Recall that D is being trained to minimize the loss, or alternatively to maximize the negative of the loss. Assume some $\hat{p}$ that represents the probability of class one given D's information state. Thus, D is being trained to maximize $\hat{p}\ln p+(1-\hat{p})\ln(1-p)$. Taking the derivative and setting to zero yields $0=\frac{\hat{p}}{p}-\frac{1-\hat{p}}{1-p}$, so $\frac{\hat{p}}{p}=\frac{1-\hat{p}}{1-p}$, $\hat{p}(1-p)=p(1-\hat{p})$, $\hat{p}-p\hat{p}=p-p\hat{p}$, so $p=\hat{p}$. Therefore, D is incentivised to output the probability that reflects its belief about the input image being real, regardless of prior. [1] This interpretability turned out to be important to our getting GANs to converge in practice, as will be expanded upon more later.

Choosing an objective for the generator is significantly more complex. The D forms a moving-target loss function whose gradient informs the generative net about how to improve. As the generative net gets better at creating realistic, artificial fills, so too does the discriminative net at detecting them, and moving to recognize new failure modes as they arise. The original GAN paper formulates this as a zero-sum, minimax game between D and G[9], and relies on this for a number of their theoretical proofs. This might suggest the generator should seek to maximize the discriminator's loss, but this is problematic for several reasons, as are briefly touched on in the paper.

Foremost among them is that it causes the gradient for the generator to collapse in the most important training case. Normally, a nice property of cross-entropy loss—in addition to the interpretability, above—is that the log's derivative cancels out the damping effect of the sigmoid. Recall that the derivative of the sigmoid function outputting p (w.r.t. its sole input) is $p(1-p)$, and that the derivative of the negative loss we are ascending is $\frac{1}{p}$ if the image is real and $\frac{-1}{1-p}$ if the image is generated. Therefore the gradient of the loss w.r.t the sigmoid's input is $1-p$ if the image is real or $-p$ otherwise. This means that the gradient feeds back in proportion to how wrong the probability output was, which is exactly what we would want; updates are larger the more wrong the network is.

Now consider what happens for a generator trying to maximize the discriminator's loss. If the discriminator has learned to be useful as a loss function, it likely outputs close to the 'correct' probability, so the magnitude of the gradient that makes it back through the sigmoid is low. This property, which is so nice for training the discriminator, destroys the gradient for the generator when it has the most to learn from the discriminator. Multiplying the gradient by a small

[1] For some reason, proofs like these seem to get less attention in CS, then Stanford's MS&E, where they have even been picked up as a probabilistic grading scheme to incentivize students to bid their true probabilities on answers.

number just as it begins propagating crushes the gradient toward zero regardless of whether the generator is taking the negative of the gradient to correct the direction.

This problem is even worse than might immediately be obvious. In the absence of the generator learning, the discriminator gets time to solidify its advantage, quickly taking its loss to near zero and bringing training to a halt.

To fight this problem, we instead have the generator minimize the loss for the opposite label from the true one. This has a nice, direct interpretation as the generator optimizing the probability of its output being classified as real. More importantly, though, it leverages the nice, sigmoid-derivative-canceling property in the opposite direction. Being able to train is well worth the extra back propagation. This also illustrates the importance of not just choosing a loss function with the right global minimum, but also beneficial gradient properties.

Before we turn to the remaining difficulties in training and the methods we used to combat them, let's put together all of the elements discussed above to describe how an GAN is trained. First, a batch of samples is loaded. In the purely generative case, these samples are noise and images from the training set. In our content-aware fill adaptation, these are masked image, original image pairs. The x's, i.e. the noise or masked images, are fed through the generator to produce generated, fake inputs for the discriminator. We then run the discriminator on both the fake and real elements of the dataset, backpropagate through the discriminator with the correct labels and the loss function, and perform an update step on the discriminator. We then backpropagate again through the discriminator using the wrong labels for just the fake examples, and propagating the error all the way back through the generator, and performing an update only for the generator. This corresponds to the alternative error function for the discriminator mentioned in the previous paragraph. We update the discriminator before updating the generator—as opposed to doing a synchronous update—because it seemed to helped training converge to visually pleasing outputs slightly more quickly. We hypothesize that this is because this lets the generator learn from the update on that particular image in addition to what the discriminator has learned previously about what is, in general, wrong with the generator's output.

For readers seeking to implement generative adversarial networks, we should note also that there are several difficulties in implementing this architecture in standard packages. In particular, you have to break out of the typical update step in Torch's optimization package to prevent updates on the second backward pass through the generator and need to invert the gradient between the discriminator and generator because frameworks tend to implicitly invert the loss gradient to turn the minimization problem into a maximization problem that can be handled by an optimization package.

5.2.2 Generating CIFAR Images

Since GANs are notoriously difficult to train, our initial effort was constructed as a proof-of-concept to attempt actually getting a GAN training. We utilized the framework from Boesen et. al [4], and applied it to our dataset. With some work, we were able to train the net and generate images from CIFAR-10 [13], shown below. We then applied our learnings and insights to our own networks.

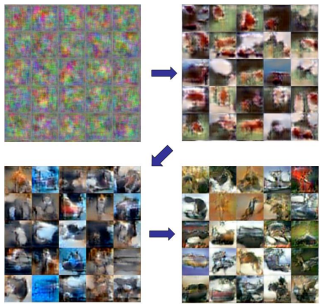

Figure 6. Images generated from our initial GAN attempt, using an input of pure Gaussian noise.

5.2.3 GAN Architecture

The final architecture for the generator was:
[64 channel 5x5 CONV + ReLU] x3
[3 channel 3x3 CONV]
And for the discriminator:
[32 channel 5x5 CONV + ReLU + Pool 2 + Dropout]
[64 channel 5x5 CONV + ReLU + Pool 2 + Dropout]
[96 channel 5x5 CONV + ReLU + Pool 2 + Dropout]
[4*4*96 channel Fully Connected + ReLU + Dropout]
[1024 channel Fully Connected]
[Sigmoid]

5.2.4 Adversarial Training Difficulties

Despite our choice of loss function, generative adversarial networks are still suffer from many difficulties in training, including the model domination problem that the loss function choice was meant to combat. If one model completely dominates the other (meaning its loss is much lower), then the model being dominated tends not to learn, or worse, the dominating model learns a brittle strategy, which the dominated model then exploits. This flips the imbalance in the other direction but results in no useful learning. The challenge, then, with GAN training is to make sure both networks learn at a relatively consistent pace: they need to "keep up" with each other.

This is often difficult to do in practice. Our first several dozen experiments attempting to train a GAN failed completely, as one network (typically the discriminator)

quickly dominated the other. We experimented with different thresholds at which we would temporarily "pause" learning for the better network (by setting its learning rate to 0), to give the worse network a chance to catch up. These thresholds were computed as the negative log of different acceptable probabilities of failure. For example, if the generator became so good at convincing the discriminator its images were real that the discriminator loss rose above $-\log(0.5)$, this would indicate that the discriminator has started having *worse than random* performance. In this case, we tell the GAN that it has exceeded its upper loss threshold, so we pause learning for the generator until the discriminator gets better. If the loss falls below a certain threshold, then we pause the discriminator until the generator catches up.

Despite many experiments with different values for these thresholds, we were initially unable to get the network to train. We eventually overcame these difficulties by removing our batch normalization layers from the generator and discriminator. Batch normalization was making these networks "too powerful", resulting in saturation and poor error signal for backpropagation. See Figure 77 for a detailed breakdown of training results with and without batch normalization layers.

5.2.5 Common Failure Modes

Although our GAN was able to generate significantly better results than our direct convolutional approach, there were still several common failure modes that would arise. Several are shown below

5.2.6 Results and Evaluation

Having a generative model means that evaluating our results is much more subtle than simply reporting accuracy on a test set, and requires more careful thought about what outputs should actually be. Moreover, a smart fill tool shouldn't really seek to duplicate the content its intended to replace. Instead, it should seek to generate a plausible alternative to the masked content. This objective is more along the lines of "indistinguishable from a real image" than it is "regenerate masked content."

Our primary approach for final objective evaluation is to simply examine the output images to see what makes them not look real to a human. Therefore, as a qualitative metric for evaluation, we examined a randomly shuffled set of generated and real images from a human perspective and attempted to classify them as real or fake. As the primary test subjects, our perspectives were biased in that we knew to look for a masked region in each of the images - however as our model improved, it became more difficult to discern if a mask existed, and where on the image it was located. In order to better quantify this metric in the future, we plan to run this test with untrained participants and measure their accuracy on the task. In the final implementations of our

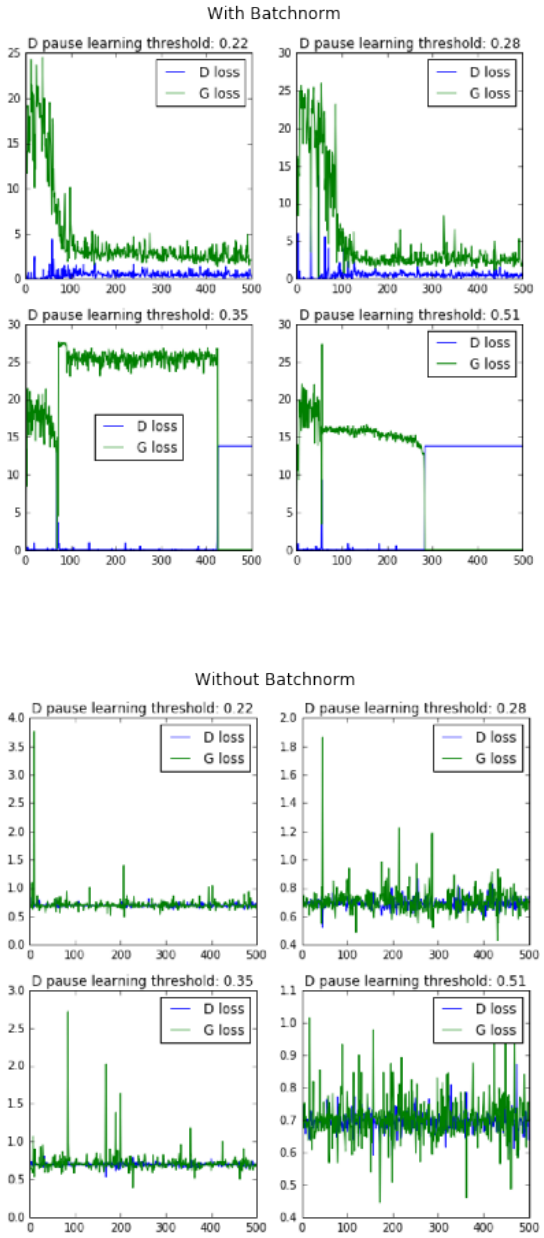

Figure 7. Training results for GANs with and without batch normalization, varying the threshold below which we pause discriminator learning. The y-axis is loss and the x-axis is 200 x (number of iterations) on each plot. Our goal is for the green and blue lines to move roughly in tandem. With batch normalization, we are unable to get the networks to train together. Note that unlike typical loss function graphs, with GANs we don't want to see losses hurtling monotonically downward. Rather, we care that the losses move together and stay consistent (or perhaps get lower) over time.

Figure 8. One common failure mode was the network producing just a similar colored chunk or smear over where the mask was.

Figure 9. Another common failure mode was the network being able to detect edges, but incorrectly interpret the overall color of the mask

Figure 10. The final common failure mode was an inability of the model to capture details in the center, and emitting a blurred result instead. We believe that this could be improved on through iterative sampling, which is discussed later

model, a set of our best images were able to fool a trained human discriminator on average 40 percent of the examples. Several of our best examples from our validation set follow — in each example, the masked area is shown on the left, the original image in the center, and the generated image on the right:

Figure 11. This image shows our network reimagining the entire front half of the frog, reorienting its front leg and providing it with two, despite having over a quarter of the image masked off.

Figure 12. Despite having significant detailed portions of the image masked off, our network was able to regenerate the idea of a rider on a horse, the rider's legs, as well as match it to the background of the image

In order to see the networks perspective and performance, we examined changes in the loss functions throughout testing and training.

For the generative adversarial net approach, we also can use the rate at which the discriminative network can distinguish filled images from unfilled ones.

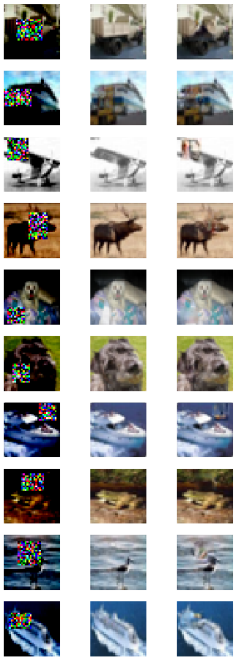

Figure 13. The above are examples of randomly generated images (not cherry-picked) from the validation dataset.

5.2.7 Convolution-Only Adversarial Network

Given our success at content-aware fill with an adversarial network using just a single fully connected layer in its discriminator, we decided to try using a fully convolutional architecture. This was motivate by the desire to start changing our architecture toward being able to handle real images of the type we might wish to edit. This requires being able to operate on arbitrarily sized and non-square inputs, which a purely convolutional approach would be able to handle. We should note that truly handling arbitrarily sized inputs would likely also require convolving over scale, as described in the Future Work section. Warping or scaling images to square is inelegant and there is reason to believe that scaling or otherwise warping damages the ability of filters to behave properly, or at least makes it harder for them to detect inputs [10]. An all convolutional architecture would avoids the need to squash or reshape such input images.

There are other reasons to prefer a convolution only model. Such a model would have very few parameters since there are no fully connected layers. Our final all convolutional model had about 6.6 times fewer parameters, for example. Furthermore, we hypothesized that it might be able to more quickly apply its learning about improper filling in one region of an image to other regions. In our previous architecture, the discriminator had trouble matching the power of the generator. The generator's learning had to be

switched off for approximately 200 of every 400 iterations to keep the discriminator from falling behind. By contrast our final all convolutional approach was much more evenly matched, with the generator and adversary each having to turn off pause training only about 5 times every 400 iterations. This led to much faster training.

Our discriminator architecture was inspired by Springenberg and Dosovitskiy, et al.'s all convolutional network[14], and uses a global average pooling layer as the last layer in the network. This layer does not currently exist in Torch, so we wrote one extending the spacial average pooling layer. We plan to submit it as a pull request in the coming weeks.

The final architecture for the generator was:
[64 channel 5x5 CONV + ReLU] x5
[3 channel 3x3 CONV]
And for the discriminator:
[64 channel 3x3 CONV + ReLU]
[64 channel 3x3 CONV stride 2 + ReLU]
[64 channel 3x3 CONV stride 1 + ReLU]
[64 channel 3x3 CONV stride 2 + ReLU]
[64 channel 1x1 CONV stride 1]
[Spatial Global Average Pooling]
[Sigmoid]

Note that using convolution removed the need for the dropout required to stabilize the fully connected layer. Also, we attempted to restrict the outputs of the generator to the space of valid images using tanh and hard tanh, but both seemed to not improve results, and slowed training dramatically.

Here are some example outputs after roughly 1,100,000 iterations:

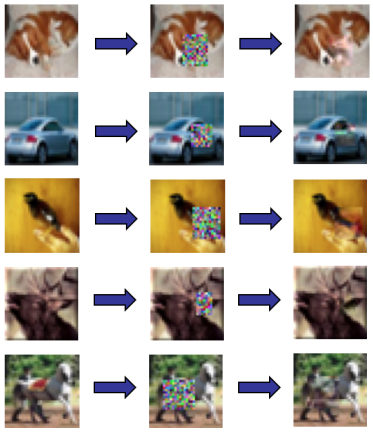

Figure 14. Several original, masked, filled triples from the all conv GAN.

The results are nearly as good as the fully connected layer with the exception of the color matching. Though it converged much more quickly, the all convolutional model never learned to perfectly blend the color of its patch with the surroundings. This can be seen in the bird, dog, and (faintly) in the car above.

6. Conclusions and Future Work

Our approach to content-aware image fill through convolutional generative adversarial networks was able to generate very plausible results for actual images. Through our experiments, we were able to confirm our hypothesis that convolutional networks are particularly well-suited to the problem of content-aware fill. Specifically, they seem to be adept at good texture detection and local focus. Combining this with a generative adversarial network allowed us to leverage the strengths of both architectures, optimizing for our final objective (image plausibility), instead of another loss function such as least-squares that would result in erroneous features (i.e. blurring) in the output image.

While we were able to obtain passable results from our direct convolutional model, they suffered from several common failure modes that our GAN was able to address. For example, our direct convolutional network outputted images where the masked region was often tinted, or in which the middle of the masked region was heavily blurred. The outputs from our GAN exhibited these flaws on occasion, but overall was able to consistently produce significantly better results. We

In terms of future work, we hope to scale up the images that we are able to apply our content-aware fill method to. To do so, we have identified several potential approaches. An issue that we anticipate encountering as we scale up is making sure that our output neurons have a large enough receptive field for larger images. To overcome this, we seek to apply the technique of spatial pyramid pooling, as described by He et al [10], but to iterate this technique at each layer. By turning each layer into a scaled pyramid, applying convolutional filters, and then reorganizing into another pyramid, we hope to get both receptive fields that grow exponentially instead of linearly with depth, and also to get convolution over scale.

Additionally, our model likely suffers from a violation of the assumption that the problem is spatially invariant. Filling the internals of regions is fundamentally different from filling edges, and should depend on how edges were filled. This suggests an iterative approach, where masked regions are filled from the outside in. Combined with pyramid mixing, we hope this will aid in obtaining plausible results over much larger images, and make GANs a scalable solution to content-aware fill.

References

[1] M. Ashikhmin. Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226. ACM, 2001.

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

[3] 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. ACM Press/Addison-Wesley Publishing Co., 2000.

[4] A. Boesen, L. Larsen, and S. Snderby. Generating faces with torch.

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

[6] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning.

[7] A. Criminisi, P. Perez, and K. Toyama. Object removal by exemplar-based inpainting. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–721. IEEE, 2003.

[8] G. Emile-Male and J. A. Underwood. *The restorer's handbook of easel painting*. New York ; London [etc.] : Van Nostrand Reinhold, 1976. Translation of La restauration des peintures de chevalet.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[11] M. J. Huiskes and M. S. Lew. The mir flickr retrieval evaluation. In *MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval*, New York, NY, USA, 2008. ACM.

[12] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[13] A. Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[14] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[15] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. 2016.

[16] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.