QR-GAN: Generating Scannable QR Images Conditioned on QR Codes

Taylor Archibald Brian Davis

Abstract

Quick Response (QR) codes are a ubiquitous mechanism for quickly conveying encoded messages. Modern publishers often try to create eye-catching, branded QR codes to promote a product or service, but are constrained by QR ISO specifications. To create more aesthetic QR codes, previous works have explored various methods of embedding a specific QR code within a pre-specified image. However, depending on the QR code and image, there may exist no acceptable solution that preserves the content of the original image while still functioning as the desired QR code. Consequently, we seek to create entirely new images that both resemble some class of images while also scanning as a QR code. This strategy adds many more degrees of freedom to the problem and should enable the creation of higher quality images that simultaneously scan as QR codes. We propose a specially tailored Generative Adversarial Network (GAN), aptly named QR-GAN, with a novel QR pixel loss that makes broad strides in realizing this objective.

1. Introduction

QR codes were developed in Japan in 1994 for the automobile industry to help inventory automotive parts. Consequently, the standards developed for QR codes are fiercely practical, including being robust to things like lighting conditions, viewing angle, occlusion, blurring, and various kinds of damage to the QR code. To accomplish this resiliency, QR codes are typically generated with maximal contrast between elements (i.e., they are black and white) and contain large, obvious anchor patterns to orient the code appropriately. With the advent of camera "smartphones," QR codes became a ubiquitous way to link users from the real world to the virtual one, and quickly became a natural extension of advertising and marketing departments. The natural tendency of any good marketing department when required to print what is effectively some monochromatic noise on their esteemed materials would be to attempt to recolor and brand the QR code the maximum extent afforded by the technology without entirely corrupting the encoded message. Individuals may also be interested in personally branding QR codes or generally improving their aesthetic.

Figure 1. A QR code image generated from the method described in [18] (left) and one from QR-GAN (right). While the first image largely resembles a specific target image, it contains obvious QR artifacts. On the other hand, besides the anchor corners, QR-GAN largely conceals these artifacts by creating a novel image.

While QR codes may seem to be poorly adapted to dramatic beautifying attempts, they are generally forgiving to minor color adjustments, provided the image still has high contrast. Moreover, because they incorporate Reed–Solomon error correction [13], up to 30% of the data can be recovered if the QR code has been corrupted. This error correction makes it fairly simple to simply drop a logo into the middle of a QR code, though significant visual alterations make it more difficult to preserve the QR content and require more advanced techniques.

Many of these techniques to embed QR codes in images or improve their aesthetic generally involve first selecting an image in which to embed some QR code. However, this places a considerable constraint on what the system can generate, since it must resemble the specified image, even if that image might be a poor candidate to embed that particular QR code from a technical perspective. Consequently, methods that follow this strategy accept certain aesthetic compromises. Often, after carefully applying several different techniques, the resultant QR-embedded image still largely resembles the original image, only occluded by a checkerboard of black and white patches that more or less resembles a QR code (as in Figure 1). This is not too surprising though, since the QR code was literally intended to make decoding as simple as possible for barcode scanners of the mid 1990's, which imposes significant constraints on what can be done to the QR code without compromising the

data. Despite this limitation, many traditional computer vision and graphical methods have achieved mixed success in aesthetically embedding QR codes in specific images.

However, recent advances in deep learning have enabled the creation of synthetic, photo-realistic images. Hence, rather than being restricted to some set images into which a QR code must be embedded, it is possible to create a model that generates an image that has been largely optimized for embedding any specified QR code. We propose such a system, the Quick Response Generative Adversarial Network or QR-GAN (*kür-găn*), which takes as input a QR code, and outputs an image that somewhat resembles a provided class of images with a scannable, embedded QR code.

It is perhaps fitting that QR codes were, in part, inspired by the game of Go; for, even as the complexity of Go is unravelled by modern deep learning, so too may be the embedding of QR codes in images.

2. Prior Work

2.1. QR Code Art

QR code artwork has been of some interest in the research community for over a decade.

Early attempts to create more aesthetic QR codes involved tweaking the QR code itself. Cox demonstrated in [4] that the QR blocks themselves could be manipulated to resemble a particular image without corrupting the content of the QR code. This was done by leveraging the one to many relationship between the number of messages and the number of valid QR code representations for that message. By changing the encoding scheme, mask, size, orientation, error correction scheme, etc., one could generate a very rough approximation given any message and image.

Halftone [2] was an early attempt that exploited the fact that QR code scanners relied primarily on the center of each QR "block" while adjacent pixels around the perimeter of the block could often be inverted. The result was still a black and white image, but one with much higher resolution and less noise, but that nevertheless scanned as a QR code. More recently, deep learning techniques have been proposed to work in conjunction with, and in place of, these prior methods. In [17], Xu et al. first perform some optimization to the QR code to be embedded, in the same vein as [4]. They then adopt a deep learning style transfer technique to render the image in the style of another image. Finally, they employ a robust error correction scheme that aims to balance the QR code readability with the visual quality of the image.

2.2. Generative methods

With the advent of deep learning, many avenues became available for synthesizing novel images. One such was the invention of the Generative Adversarial Network (GAN) [8]. In this framework, two neural networks are used: the *generator*, which seeks to synthesize images of a certain class, and the *discriminator*, which attempts to discern whether an image was created by the generator network or drawn from a set of ground truth (GT) images of that same class. They are "adverse" to each other in the sense the generator's loss is minimized when it deceives the discriminator, and the discriminator's loss is minimized when it successfully distinguishes the generator's synthetic images from the GT images. The loss function is such that the discriminator essentially provides feedback in the form of backpropagated gradients to the generator to create better synthetic images, while the discriminator learns from the synthetic and GT images it misclassifies.

Many kinds of conditional GANs followed, including pixel-to-pixel generator (Pix2Pix) [9]. Pix2Pix leverages the GAN framework on pairs of images with shared content but different styles or classes. The Pix2Pix GAN is conditioned on one image in the pair and trained to generate an image of the other class, relying on the content of the first image and learned representations of the target class.

Recent GANs can also successfully model the content and style of an image. StyleGAN [11] and the improved StyleGAN2 [12] learn a mapping from both random noise and style vectors that influence style by controlling the generator network feature maps via AdaIN layers.

3. Method

Our goal is to produce images which will scan as QR codes, but which mimic the class of images in the training dataset.

We use a conditional pixel-to-pixel generator [9], trained with two losses, one to encourage characteristics of the training set (adversarial) and one to encourage the properties that make an image a valid QR code (reconstruction).

QR-GAN is based on a modified StyleGAN2 [12]. Specifically, the generator is modified to accept an image input, while the discriminator is largely unchanged, but for the addition of a specific pixel-level mask, described in 3.2.

The model is trained to generate images based on QR codes of a specific size or version. To train for a particular version, the various masks must be adjusted accordingly.

3.1. Generator

The generator in QR-GAN accepts a 256×256 image QR code and outputs a new 256×256 image, which should decode with the same message as the input QR code, but is changed to mimic a specific set of images in some way. StyleGAN2 follows a pattern of (style) convolutional blocks and up-sampling, starting from a learned constant 4×4 feature tensor and ending at the desired resolution. In our case, we up-sample to 256×256. We modify the architecture to have a down-sampling CNN branch, which reduces the

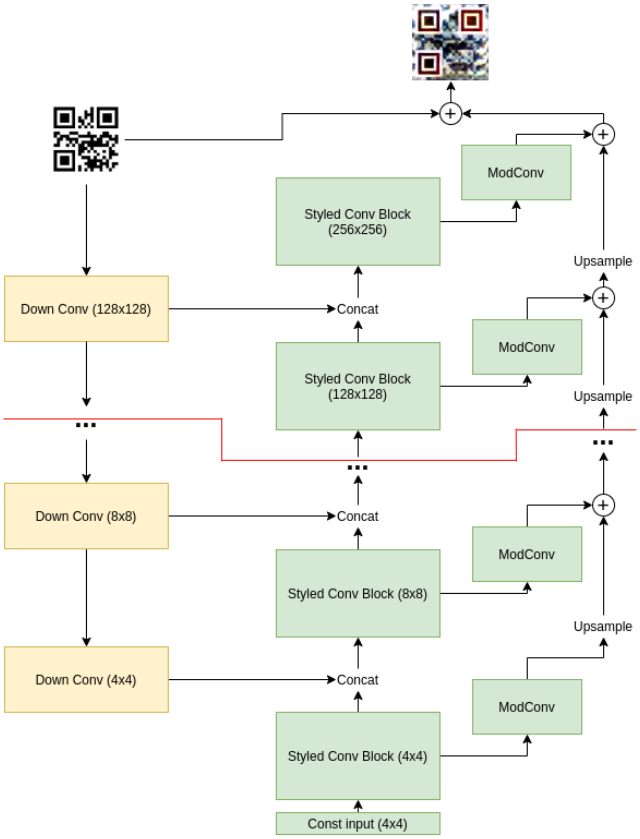

Figure 2. Our generator architecture. The green is StyleGAN2. We predict an offset from the input QR code image.

dimensions down from 256×256 to 4×4. The input QR code (image) is passed into this down-sampling branch, and features taken from each resolution are appended to features in the (original) up-sampling branch at the matching resolution. This resembles a UNet [14] architecture, but still includes the starting 4×4 feature tensor that StyleGAN2 uses. Instead of having it directly predict pixels, our model predicts an offset from the input QR code, such that the network output is added to the QR image. The output of the network is unconstrained (as in [12]). Our generator is shown in Figure 2. The green boxes represent the original StyleGAN2 architecture.

3.2. Discriminator

If QR-GAN is producing functional QR images, the distribution of generated images will not resemble the training dataset distribution. While the QR code binary information can be represented in a way that more closely resembles the distribution of light and dark sections of the target class of images, the QR anchor points (bulls-eye pattern in corners) are fixed and cannot move (and we do not allow rotation). This makes the discriminator's task almost trivial, as it needs only learn to recognize the anchor points. The loss

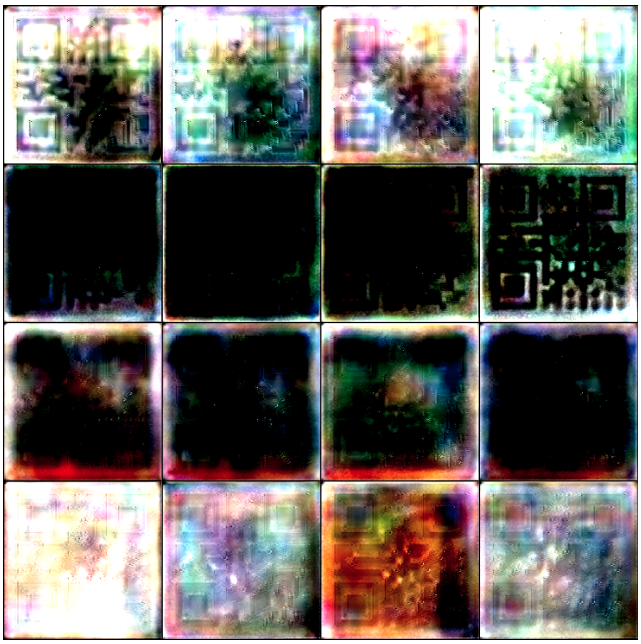

Figure 3. Results without discriminator masking. Each row is a batch, each batch being generated about 5000 iterations apart. The generator attempts to hide the distinctive QR anchors with large color and contrast shifts.

coming from the discriminator will be directly competing with the loss encouraging a valid QR code (as it encourages the anchors at a pixel level). This leads to unstable behavior as seen in Figure 3. To alleviate this issue, we occlude the portion of the image where the anchor patterns would appear in the QR code image before passing the image to the discriminator. This means these pixels are never supervised by the adversarial loss. Often, the generator creates anchor patterns with colors consistent with the rest of the image, perhaps due to its convolutional nature. While this is not always the behavior, the anchor points are nevertheless perhaps the most highly constrained portion of the QR code, and we opt to focus our efforts on the portions of the image over which we can have the greatest influence. We note that [17] simply paste the anchor pattern onto their generated images.

Our discriminator architecture largely mirrors that of StyleGAN2 [12], with the addition of corner masking.

3.3. Losses

The adversarial loss is the same as the one employed in StyleGAN2 [12], and the generator and discriminator regularizations remain the same.

We include a second loss, a masked QR reconstruction loss, which encourages the important elements of a valid QR code. This acts at a pixel level and does not quite capture the nuances of what makes an image a valid QR code. Because the magnitude of the losses differ, we balanced the

gradients from the losses in a manner similar to [5].

We also attempted an approach which mimics those commonly used in handwriting generation ([1, 10, 7, 5], which uses the loss flowing from a handwriting recognition network to inform the generator how to generate legible handwriting images. Our attempt involved training a QR decoder network, which predicted both the message of a QR code image and whether another QR decoder could read it. We hoped this would provide detailed information to the generator about which pixels needed to be changed to make generated images function as valid QR codes. We were unable to get better results using this method. We suspect the decoder network was not trained robustly enough to provide the needed feedback.

Another reason the QR decoder may not have helped is because it was not sufficiently dynamic. The success of a GAN largely depends on the discriminator evolving with the generator. If the discriminator is frozen early in the training, the generator would quickly learn certain noise patterns that exploit the discriminator, while not producing images that sufficiently resemble the target class. Similarly, the generator network might be able to quickly learn to exploit the QR decoder network without generating valid QR codes, while subsequent training primarily improves the generator's loss from the discriminator. While we attempted dynamically training the decoder with images from the generator, this does not fully resolve the issue, and may suffer from being a bit of a feedback loop.

While the handwriting decoder was helpful in [5] despite being static, we note that those losses were generally aligned in their aims: creating legible handwriting. In our case, our losses are largely opposed: the intersection of the classes of images we tested and QR codes is virtually nonexistent. Hence, as the discriminator coaxes the generated images to resemble, e.g., abstract art, the masked QR reconstruction loss compels the generated images to resemble QR codes, i.e., images that typically look nothing like abstract art.

3.3.1 Masked QR reconstruction loss

One could apply a pixel-wise loss between the original QR code images and the generated one to encourage the generated image to be a valid QR code. However, many of the pixels of a normal QR code are not important in decoding. For example, most QR decoding algorithms only use the center pixels in each data cell (square region that is either off or on). We incorporate this knowledge into our loss.

To compute the loss, we first compute an L1 hinge difference between the QR code image and the generated image. We then multiply this difference image by two masks. One mask has square regions covering the corner anchors, including a padding of 1 cell. The other mask has a value of 1

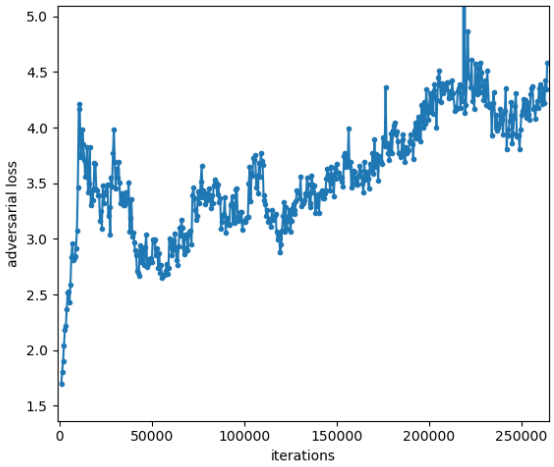

Figure 4. Adversarial loss during training.

at the center pixel of each data cell, a value of 0.5 in the 3×3 area around this and 0.25 in the 5×5 in the area around this, minus corners (more circular). This informs the generator that it is the center pixel which most carry the information, the other pixels can be colored in variable ways. The result of each mask multiplication is summed and divided by the sum of the respective mask (so each part of the loss is balanced). The mean of these two values is the final loss.

3.3.2 Loss balancing

We balance the adversarial and masked reconstruction loss like [5] where the gradients from the QR reconstruction loss are normalized to match the gradients from the adversarial loss. We used balancing weights of 1.0 for the adversarial loss and 1.4 for the reconstruction loss. We place more weight on the reconstruction loss to prioritize the creation of valid QR codes over the realism of the generated image.

Gradient balancing is important as both losses vary greatly, as seen in Figures 4 and 5. Without balancing, whichever loss was highest would dominate because of their competing objectives. Gradient balancing is also useful as it automatically balances the gross differences in magnitude.

4. Training

We train using mostly the same hyper-parameters as StyleGAN2 [12]. We use a batch size of 32. We train about 300,000 iterations, although some of the models were cut short as they were not improving.

Because of the competing nature of the two losses, the model's ability to generate valid QR codes fluctuates rapidly. To ensure we can generate valid images consistently, we track the number of images which can be decoded

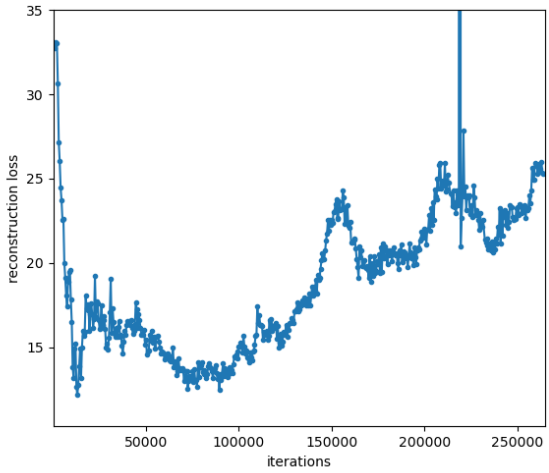

Figure 5. Masked QR reconstruction loss during training.

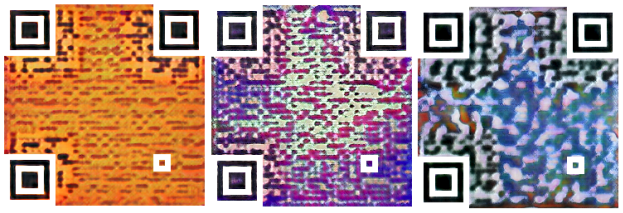

Figure 6. Results from a model with less weight on the reconstruction loss than the adversarial loss. While these variously resemble abstract art better than other examples, the vast majority of images generated from this model would not scan without post-processing.

(by a QR decoder). A snapshot of the model is saved when the model is generating at least 90% valid QR images for the batch.

Because the QR decoder used to validate our QR codes during training does not perfectly mimic typical QR decoders found on smartphones, our snapshots only generate smartphone-scannable QR images about 25-50% of the time. In practice, because it is very easy to check if a QR code is valid, a user might generate many different candidate images for the desired QR code, with invalid QR code images being pruned by the computer automatically. The user can then select the image that best conforms to their taste from the images with valid QR codes.

While worse snapshots (one with a lower ratio of valid QR images) seem to generate higher quality images (Figure 6), we emphasized usability (not having to test many images to find a valid one) in our experiments.

5. Results

We test our method on several datasets. All images were resized to 256×256. We used QR code version 1 (22×22 cells) with the lowest error correction ("L"). Note that this lower level of error correction variously makes generating images valid QR codes more difficult, since there is less redundancy. Increasing the level of error correction may allow for the creation of more realistic images that still scan as QR codes.

We sampled random strings from length 4 to 17 of valid URL characters and generated QR codes using the default QR masking. The results shown were generated using a snapshot as described in the previous section, and are hand-picked to demonstrate variety of valid QR code images.

Figure 7. Example images from abstract art dataset.

Figure 8. Generated images from model trained on abstract art dataset.

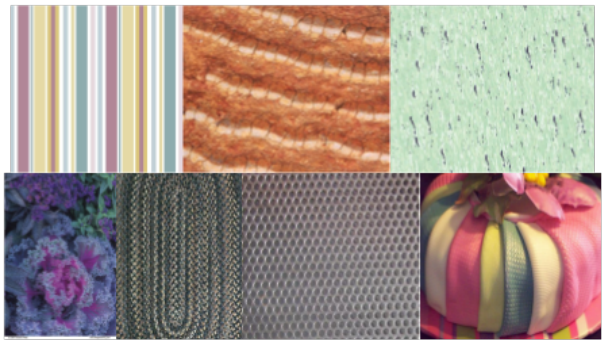

Figure 9. Example images from the Describable Textures Dataset.

Figure 10. Generated images from model trained on texture images.

5.1. Abstract art

We collected a variety of images of abstract art (examples in Figure 7). These are mostly paintings, but there are also some photographs. We felt that some structures seen in abstract art might be able to hide the QR code data. Results can be seen in Figure 8.

5.2. Textures

The Describable Textures Dataset [3] contains photographs of a variety of textures (examples in Figure 9). As a QR code is a particular binary texture, we though it may learn to adapt other textures to match its pattern. Results on this dataset can be seen in Figure 10. This dataset had the most success at generating images which, apart from the anchors, did not look like viable QR codes, but still successfully scanned.

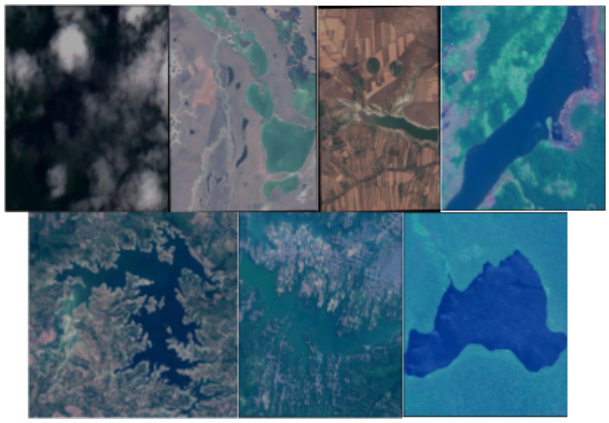

Figure 11. Examples from the Satellite Images of Water Bodies dataset.

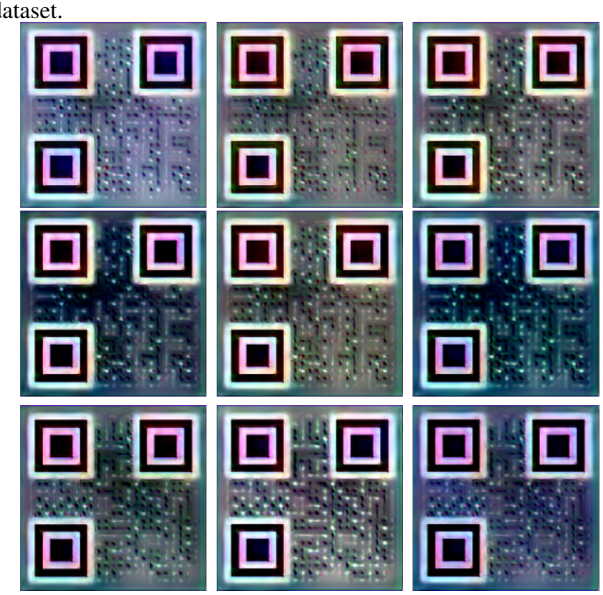

Figure 12. Generated images from model trained on satellite images. The model only captures color information from the dataset.

5.3. Satellite images

We also trained QR-GAN on the Satellite Images of Water Bodies dataset [6] (examples in Figure 11). Results on this dataset can be seen in Figure 14. While we hoped the model would produce lake- or river-like QR images, it fails to learn any of the structure from the dataset, and only mimics the color palette. This dataset may have lacked variety, making the discriminator's job too easy.

5.4. African fabrics

The African Fabric Images dataset [15] contains images of patches of fabric patterns (examples in Figure 13). This dataset was chosen for much of the same reason as the texture dataset. Results on this dataset can be seen in Figure 14. Because images in this dataset had low resolution (64×64) and had to be up-sampled to 256×256, the gen-

Figure 13. Examples from the African Fabric Images dataset.

Figure 14. Generated images from model trained on African fabric images. Because the fabric images are low resolution, the generated images contain pixel artifacts.

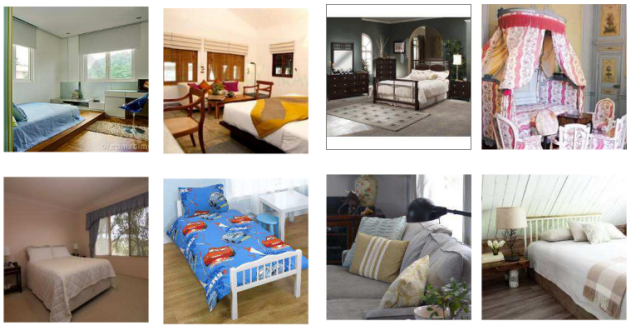

Figure 15. Example bedroom images from the LSUN dataset.

Figure 16. Generated images from model trained on bedroom images from the LSUN dataset.

erator learned pixel artifacts.

5.5. LSUN bedrooms

The LSUN dataset [19] contains photographs of scenes with 10 labels. We only use bedroom images (examples in Figure 15). While the models trained on the other datasets potentially could generate a valid QR images which roughly resembled the data distribution, this would be virtually impossible for natural scene images. However, we still wanted to test its performance on such a dataset. Results for LSUN bedrooms can be seen in Figure 16. We were surprised that the images do resemble the early stages of training a GAN on LSUN bedrooms, albeit blockier, as the generator attempts to produce viable QR codes.

6. Conclusion

We have presented QR-GAN, a method of training a generator that accepts a QR code as input and outputs novel images that decode as a QR code containing the original message. We employ GAN training along with a masked QR reconstruction loss, using a modification to the StyleGAN2 architecture. We can generate valid QR images frequently enough that a suitable one can be hand selected by a user for an arbitrary QR code representation. While the generator does not create images satisfyingly indistinguishable from training dataset, the training dataset nevertheless strongly influences the type of images generated.

Our method is somewhat flawed as the masked QR reconstruction loss does not guarantee convergence to a generator that produces valid QR image. A supervising QR decoder network that is trained with generated images, much in the same way the discriminator is, should be able to achieve this. The decoder supervision would likely have many benefits to the model. However, we were not able to get this to work given the time constraints we had.

While many of the generated images decode as QR codes, they are often difficult for QR detectors to locate. StegaStamp [16] demonstrates that robustness can be induced by including deformation and noise introductions. This would be most beneficial with supervision from a QR

decoder network.

While we conditioned our generator on an arbitrary QR code, QR codes can be manipulated to vary in appearance, but contain the same message. Xu et al. [17] perform this as a preprocessing step in their QR image generation method. This could similarly be applied at the beginning of our method, perhaps learning what QR code arrangements are the most amenable for generating images of the target class.

References

[1] Eloi Alonso, Bastien Moysset, and Ronaldo Messina. Adversarial generation of handwritten text images conditioned on sequences. In *Int. Conf. Document Analysis and Recognition (ICDAR)*, 2019. 4

[2] Hung-Kuo Chu, Chia-Sheng Chang, Ruen-Rone Lee, and Niloy J Mitra. Halftone qr codes. *ACM Transactions on Graphics (TOG)*, 32(6):1–8, 2013. 2

[3] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *CVPR*, 2014. 6

[4] Russ Cox. Qart codes posted on thursday, april 12, 2012., Apr 2012. 2

[5] Brian Davis, Chris Tensmeyer, Brian Price, Curtis Wigington, Bryan Morse, and Rajiv Jain. Text and style conditioned gan for generation of offline handwriting lines. In *BMVC*, 2020. 4

[6] Francisco Escobar. Satellite images of water bodies v2. https://www.kaggle.com/franciscoescobar/satellite-images-of-water-bodies. Accessed: Dec 2020. 6

[7] Sharon Fogel, Hadar Averbuch-Elor, Sarel Cohen, Shai Mazor, and Roee Litman. ScrabbleGAN: Semi-supervised varying length handwritten text generation. In *CVPR*, June 2020. 4

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2

[9] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017. 2

[10] Lei Kang, Pau Riba, Yaxing Wang, Marccal Rusinol, Alicia Fornés, and Mauricio Villegas. GANwriting: Content-conditioned generation of styled handwritten word images. In *ECCV*, 2020. 4

[11] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 2

[12] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, 2020. 2, 3, 4

[13] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960. 1

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Int. Conf. Medical image computing and computer-assisted intervention*, 2015. 3

[15] Ayomikun Samuel. African fabric images v1. https://www.kaggle.com/mikuns/african-fabric. Accessed: Dec 2020. 6

[16] Matthew Tancik, Ben Mildenhall, and Ren Ng. Stegastamp: Invisible hyperlinks in physical photographs. In *CVPR*, 2020. 7

[17] Mingliang Xu, Hao Su, Yafei Li, Xi Li, Jing Liao, Jianwei Niu, Pei Lv, and Bing Zhou. Stylized aesthetic qr code. *IEEE Trans. Multimedia*, 2019. 2, 3, 8

[18] Zhe Yang, Yuting Bao, Chuhao Luo, Xingya Zhao, Siyu Zhu, Chunyi Peng, Yunxin Liu, and Xinbing Wang. Artcode: preserve art and code in any image. In *Proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing*, pages 904–915, 2016. 1

[19] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop, 2016. 7