Quantized GANs for Mobile Image Reconstruction

Andrew Deng
Stanford University
andrewde@stanford.edu

Wenli Looi
Stanford University
wlooi@stanford.edu

Alex Tsun
Stanford University
alextsun@stanford.edu

Abstract

We tackle the problem of reconstructing images with more than half of the original pixels damaged using a compressed, quantized generative adversarial network (GAN) designed to fit in the limited storage capacity of mobile devices. Modern work on this topic has been relatively successful using generative adversarial networks, and so our approach relies on these as well. For computational efficiency in both time and space, we experimented with reducing the precision of our weights, even to the extreme of binary weights. Our training dataset was comprised of front-facing celebrity images, which were then corrupted (by us) as input. The desired output of our model would be the original image. With 64x64 images, we achieved a final quantized model size of 3.2 MB while generating realistic images (if not very similar to the original) a majority of the time, as you can see in the results. We also show the reconstructions for our "lighter" (lower precision) models in comparison to see the tradeoff between efficiency and quality. An interesting future direction is to explore patch-based discriminators - discriminators that can identify local regions where the image is fake.

1. Introduction

Image reconstruction is an increasingly interesting subfield of computer vision, which includes super-resolution and restoration. With the rise of deep convolutional neural networks and computational resources, nontrivial methods for doing such tasks have arisen. More recently, the idea of generative adversarial networks (GANs) has given us even better results. A GAN consists of two neural networks: a generator which attempts to create new images from some distribution, and a discriminator which attempts to determine if an image was real (from the dataset) or fake (from the generator). These two networks are pit in a two-player zero-sum game in order to improve each other.

Our task was to tackle image reconstruction in a space-efficient manner. Sometimes, we have images that are corrupted in some way and would like to see what should replace the missing or corrupted pixels. For example, your friend could have taken a blurry picture for you, and you want to make it clearer. Or, if you broke up with someone recently and want to remove them from it while keeping the background, this could work as well. Efficient image reconstruction would be especially useful on mobile devices. For example, you may have taken a picture without realizing that your camera lens is dirty and thus want to reconstruct it on your phone to show others. Thus we experimented with quantization as a method to reduce the model size and make it more usable on mobile devices.

The input to our algorithm is a 64x64 corrupted RGB image of a face. The corruption "type" we applied when training was randomly selected from the following:

1. Partial occlusion by a randomly placed, randomly sized, white rectangle.
2. White noise in random locations (determined independently), corrupting more than half the pixels in expectation.
3. RGB noise in random locations (determined independently), corrupting more than half the pixels in expectation.

We then use a deeply convolutional generative adversarial network (DC-GAN) to output a 64x64 recovered RGB image.

Currently, we are training on a dataset of celebrity faces (described further in section 4), and are trying to recover the original face after noise is injected. However, this could be extended to other tasks by training on other datasets. We could remove censorship from videos or pictures, or reconstruct backgrounds if someone or something undesirable is in the picture.

When we add random noise everywhere (the second and third types), we are essentially trying to learn how to interpolate missing pixels scattered across. This is how resizing images works. For example, if we take a 2x2 image and want to make it 4x4, we put each of the original pixels into the top left corner of each of the 4 2x2 boxes, and interpolate the rest. One classical approach for resizing is bilinear interpolation - our approach would allow for more complex nonlinear interpolations.

The noise we add severely corrupts the image. A majority of the time, we corrupt at least half of the image by replacing pixels with either white or a random color. The white box noise is especially difficult, because there are exponentially many ways to fill in the missing pixels, and there aren't any closeby neighbors to interpolate from (like in the other types of noise we added). The generator must somehow recover an image that is the "most realistic" so that the discriminator loss is maximized (by thinking the generated image is real).

Another challenge is that training GANs is challenging with large images due to immense resources required. Even with 64x64 images that we are using, training the model required several hours to produced acceptable results and exhausted all of the Google Cloud credits provided in a short period of time. Though we are able to make good progress here on our relatively small images, training the model with HD phone images (e.g. 3000x4000) would require a much larger amount of resources.

Given the recent explosive interest in image-focused social media such as Snapchat and Instagram, we experimented with reducing the model capacity and precision in an effort to make these models accessible to smartphones. We experiment with various precisions to balance the tradeoff between quality and effiency.

2. Related Work

There are several fairly recent works that have found success in using GAN-based models for image reconstruction. Liu et. al. [4] designed a method called X-GANs that can reconstruct images that have been corrupted by extreme levels of noise. In their method, the generator is paired with a multi-scale discriminator that can capture feature details at different scales. Additionally, the standard adversarial loss is augmented by feature matching loss functions, which draw from trained weights in both the discriminator and another pre-trained network (VGG16). The resulting model produces realistic looking reconstructions of images, though in cases where the model must inpaint a large continuous region the produced image can differ from the original image. We base our model primarily on this work, albeit without the multi-scale discriminator and replacing VGG with SqueezeNet because of resource limitations.

Another work by Nazeri et al. called EdgeConnect decomposes the task of image inpainting into two segments [6]. First, one generator is trained to reconstruct a black-and-white edge map of the image, then another generator takes as input the produced edge map and the original RGB corrupted image to produce the final reconstructed RGB image. We attempted this edge-based approach, however it produced poor results for inpainting on large regions and it did not appear to work well for the white noise and RGB noise, both of which were not attempted by Nazeri et al.

A third method called PatchGAN by Yuan et al. [13] uses a generator paired with both a global discriminator and local discriminators that look at specific patches. It is also paired with an edge process function to further guide the generator with semantic knowledge from the uncorrupted parts of the image. We did not attempt this method due to time constraints.

Various other works were incorporated to improve some miscellaneous parts of our model. In [11], the authors found that replacing batch normalization layer with instance normalization layers improved the performance of a GAN trained for style transfer. We incorporated this work by simply replacing all of our batch normalizations in the generator with instance normalizations.

A work on dilated convolutions [12] found that using dilated convolutions improved the ability of a semantic segmentation model to understand multiscale contextual information. Since our generator should also be able to recognize features of the corrupted image at different scales we incorporate this into the residual blocks of our generator.

A paper on deconvolution (transpose convolution) [7] found that using transpose convolutions to upsample activations caused outputs to develop odd checkerboard-like artifacts. It was concluded that replacing transpose convolutional layers with a simple nearest-neighbor upsample layer and another convolutional layer produced superior results. In our work we tried both, however we failed to notice a significant difference in the outputs.

For the second part of our project, we investigated reducing the size of the generator model through weight discretization to make it more usable on mobile devices. This part of the project was primarily motivated by an older work on binary networks called XNOR-Net [9]. This work created a modified AlexNet classifier with both binary weights and binary inputs/activations and achieved close accuracy to the original AlexNet. The significant performance and memory improvements of the modified network inspired discretization as a means to achieve better computational performance. However, most of the works on discretization in existence study the effects on classifiers, so we were motivated to instead test the effects on generative models.

Later work by Tang et a.l [10] achieved higher accuracies and smaller model sizes than XNOR-Net through several improvements. A key insight they had was that L2 regularization used by XNOR-Net and others only tries to minimize the magnitude of the weights but alternative forms of regularization are more effective in making the network more suitable for quantization. We use their suggested regularization method in our model.

3. Methods

For the image reconstruction task, we designed and implemented a GAN using PyTorch [8] consisting of separate convolutional neural networks for the generator and discriminator. The generator is built to generate a 3x64x64 image output when given a 3x64x64 input. In this use case, the input to the generator is the corrupted image and the output is the model's prediction of the original image. The discriminator is designed to differentiate between images reconstructed by the generator and real images from the original dataset. These networks are pit against each other in a zero-sum game so that the generator learns to better fool the discriminator, and the discriminator learns to differentiate between the real and fake images better. At a high level, the minimax objective function can be expressed as follows:

$$\min_{\theta_g} \max_{\theta_d} \Big[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x)$$

$$+\mathbb{E}_{x\sim p(z)} \log \left(1 - D_{\theta_d}(G_{\theta_g}(z))\right)\Big]$$

Here, the discriminator tries to maximize the objective of being able to distinguish real and generated images, while the generator tries to minimize the objective so that it can fool the discriminator. In our case, $p(z)$ is the distribution of corrupted images that we generated, while p_{data} is the distribution of original images. D maps an input image to likelihood of it being real, and G maps a corrupted image to a predicted reconstructed image.

The architecture for the generator (Figure 1) was based on the architecture used in multiple image reconstruction papers: X-GANs [4] and Patch-GANs [1]. At a high level, the architecture begins with a series of convolutional blocks (Figure 3(a)) that reduce the width and height of the activation volume while increasing the depth. This downsampling section is then followed by a series of residual blocks, with inputs and outputs kept at the same size. Finally, following the residual blocks are another series of transposed convolutional blocks (Figure 3(b)) that increase the size of the outputs to reach the same size as the original input.

The downsampling network consists of 2 convolutional blocks, each of which reduces the width and height dimensions of the input by a factor of 2. Each block is made of a convolutional layer with 5x5 filters, padding 2 and stride 2. The stride of 2 accomplishes the desired downsampling. After the convolutional layer is an instance normalization layer [11]. This layer normalizes the outputs of the convolutional layer before the activation layer. Instance norm is used rather than batch normalization as the authors of [11] claim that it works better for stylistic tasks such as image reconstruction and inpainting. Following the instance normalization is a standard ReLU layer. The first block has a filter depth of 64 and the second block has a filter depth of

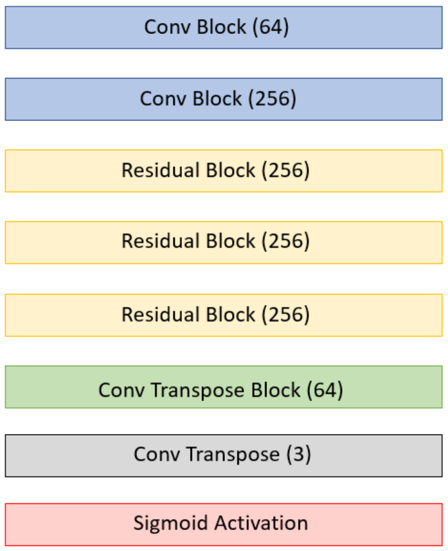


Figure 1. Generator Architecture

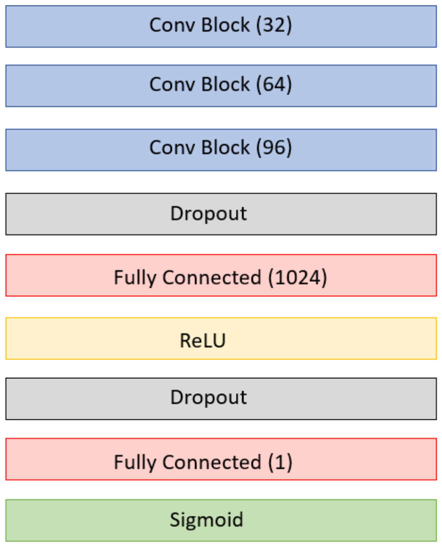
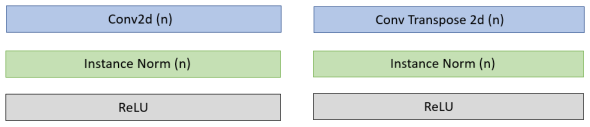


Figure 2. Discriminator Architecture



(a) Conv Block (b) Conv Transpose Block

Figure 3. Conv and Conv Transpose Blocks

256. This produces an activation volume of 256x8x8 after the downsampling network.

Each residual block (Figure 4) is made of two convolutional layers with a skip connection. The input is passed through a convolutional layer of filter depth 256 with 3x3

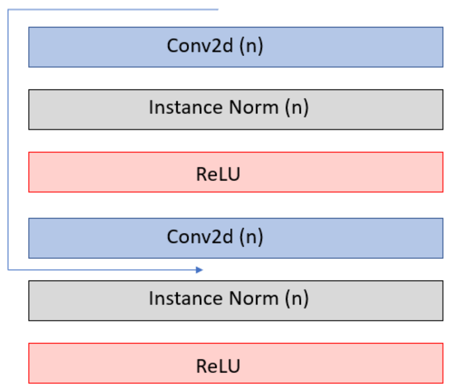


Figure 4. Resid Block

filters, which is padded so the activation size does not change. It then passes through a normalization layer, a ReLU, and another convolutional layer with the same size as the first. The original input is then added to the intermediate activation (which is the same size because the convolutional layers did not change any of the dimensions). Finally the combination is passed through another ReLU and output.

The upsampling network is a reversed version of the downsampling network; two transpose convolution layers are used, with each layer increasing width & height by a factor of 2. Between the two transpose convolution layers are another instance normalization layer and another ReLU. At the end of this subsection is a sigmoid activation to place the dynamic range of the output between 0 and 1.

The discriminator (Figure 2) used is a very standard CNN classifier that predicts images as belonging to one of two classes, real or fake images. It consists of 3 convolutional blocks, each of which is made of a convolutional layer (filter size 5, padding 2), a max pooling layer (size 2, stride 2), a ReLU layer and a dropout layer (prob 0.2). Dropout helps prevent the discriminator from overfitting. The blocks had filter depths of 32, 64, and 96 respectively. After the convolutional blocks were a pair of linear layers and a Sigmoid activation.

To train the network, we alternated between training the discriminator and generator. The discriminator's loss function was set to the standard binary cross-entropy loss to improve its ability to classify between real and fake images.

$$\mathcal{L}_D(x) = -\log\left(D(x)\right) - \log\left(1 - D(G(x))\right)$$

In the expression above, $D(x)$ represents the response of the discriminator to an input x and $G(x)$ represents the output of the generator in response to an input x. $D(G(x))$ then represents the response of the discriminator to the output of the generator.

The generator, on the other hand, requires a more complex loss function to generate somewhat realistic images. There are four weighted components of the overall loss function for the generator:

1. The first component of the loss function is the discriminator adversarial loss. This component measures the ability of the generator to fool the discriminator into mispredicting. Therefore minimizing this component should result in generated images that are hard to distinguish from real images in the training set.

$$L_1 = \log\left(1 - D(G(x))\right)$$

2. The second component is the feature matching loss from the discriminator. The feature matching loss is the ℓ_1 norm of the difference between the intermediate activations of the target (original) image, and the generated image. We produce the intermediate activations after each convolutional layer in the discriminator for both the real image and the generated image, since these activations correspond to low-level features recognized by the discriminator as useful in discriminating between real and fake images. This encourages matching features the CNN found (such as nose shape, eye type, etc), and not penalizing being off by some color.

$$L_2 = \sum_{i=1}^{N} \|F^{(i)}(x) - F^{(i)}(G(x))\|_1$$

In the equation above, $F^{(i)}(x)$ represents the intermediate activation of the ith layer of the discriminator to the input noisy image x. $G(x)$ represents the output of the generator in response to x.

3. The third component is the feature matching loss from a pre-trained network that was trained on ImageNet. Initial versions of our algorithm used VGG16 as a pre-trained network, but size limitations led us to use SqueezeNet 1.1. The intuition for using a network trained on ImageNet is that since the network was trained on such a comprehensive dataset, the features it learned should be fairly well-structured features.

$$L_3 = \sum_{i=1}^{N} \|S^{(i)}(x) - S^{(i)}(G(x))\|_1$$

In this equation $S^{(i)}(x)$ represents the intermediate activation of the ith layer of Squeezenet 1.1 in response to input x.

4. The final component of the loss is the direct pixel-matching loss. This is just the ℓ_2 norm of the difference between the target and generated image. This encourages the generator to keep the known parts of the image

the same, so the uncorrupted parts are not distorted too much.

$$L_4 = \|x - G(x)\|_2^2$$

The overall loss is

$$\mathcal{L}_G = \lambda_1 L_1 + \lambda_2 L_2 + \lambda_3 L_3 + \lambda_4 L_4$$

with $\lambda_1 = 1, \lambda_2 = \lambda_3 = 0.00001$, and $\lambda_4 = 0.001$.

In addition to just training the generator to reconstruct images, we also experimented with how reducing precision in the weights of the generator would affect the quality of the output. A reduced precision model for the generator would be beneficial in both storing the model on a device with very little storage space, and smaller weights could be exploited to improve speed. To experiment with this we trained the model in full precision, with an added loss function to encourage weights closer to -1 and 1. This function was

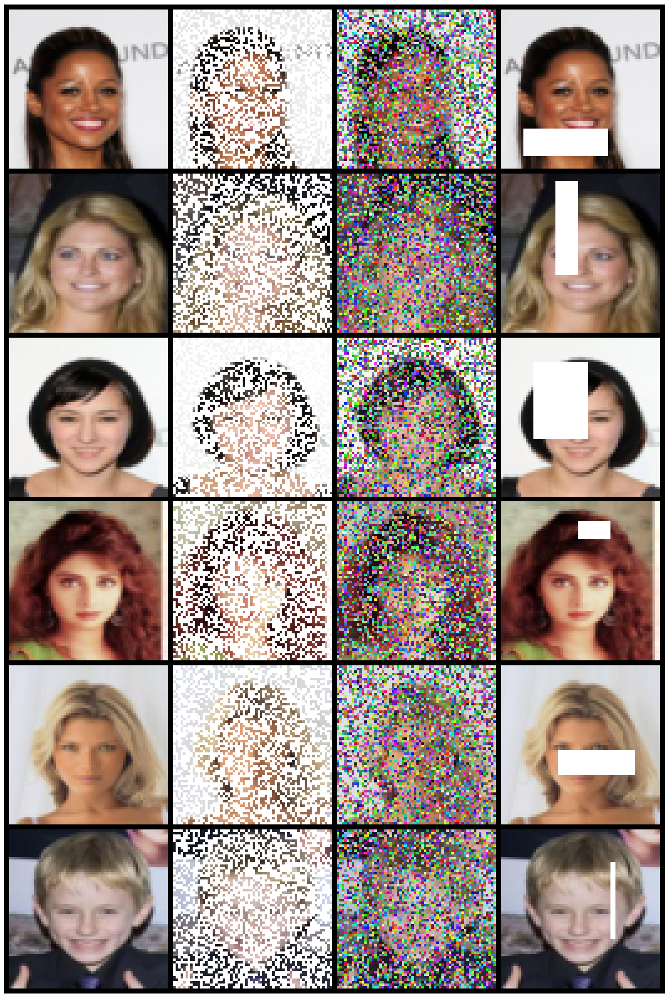
$$L_5 = \sum_{w \in P} \left(1 - w^2\right)$$

where P is the set of all learned parameters in the model. This was done as suggested by [10] which had good results using this regularizer. We add $\lambda_5 L_5$ to $\mathcal{L}_G$ with $\lambda_5 = 0.0001$ in this case. After saving the trained weights, we quantized them at various levels of granularity and computed PSNR and compared the qualitative results.

4. Dataset

The dataset we are using is a set of images of celebrities' faces from a Kaggle competition. It is called "Large-scale CelebFaces Attributes (CelebA) Dataset" [5]. There are 202599 images of faces, all of size 3x218x178. For training, we had 20,000 images, and for validation and testing, 1000 images. We didn't use all the images due to the high resource consumption needed to train these models. Instead, we opted to try out several different types of models. Each image is centered and aligned so that the person's face appears in the center vertically. Additionally, each image comes with labels on 40 different attributes describing various facial features, but these were not used in this project. We didn't perform any data augmentation, nor normalization. Additionally, as a preprocessing step, the images are resized to 3x64x64 for input to the network. The reduced image size allows the task to be accomplished with a smaller network and less training time, which helps with our limited time frame and resources.

To generate noisy images as input to our generator, we apply 3 different kinds of corruption to our images. In type 1, we uniformly sample pixels to keep with a fixed probability and set all other pixels to white. The sample rate was initially set to 0.6, but was decreased to 0.4 in order to make the task more difficult. This form of noise results in a very sparse representation of the image, with only the general shape of the person's face perceivable by the human eye.



First column: original image
Second column: corrupted by white noise
Third column: corrupted by RGB noise
Fourth column: corrupted by random white rectangle
Figure 5. Different Noise Types

In type 2, we similarly uniformly sample pixels and set all other pixels to a random RGB color. The sample rate is kept the same as in type 1. The noisy images appear more heavily distorted than the ones in type 1 to the human eye, since pixels are assigned random colors. However numerically the data is not that different from type 1 so it should not be too much more difficult for the network to handle.

In type 3, we generate 2 random pixel coordinates that create a valid rectangle on the image. All pixels within this rectangle are set to white. This creates a blank space for the network to fill in from scratch. Therefore the network must learn to draw an appropriate replacement for the missing section given the context in the rest of the image.

5. Experimental Results

Hyperparameter tuning was a key part of our experiments. Overall, we tuned the hyperparameters by training the model with a given set of hyperparameters and evaluating the model's performance on the validation set.

Both qualitative and quantitative evaluation was used. For qualitative evaluation, we took a look at the reconstructed images and observed if there seemed to be a noticeable change in image quality. Quantiative evaluation was performed by calculating the average Peak Signal-to-Noise Ratio (PSNR) of the reconstructed images before and after the hyperparameter change. PSNR is widely considered to be a good quantitative measure of human perception of image quality in reconstruction tasks [2].

Adam was chosen as the optimizer as it was found to work well with minimal hyperparameter tuning. With a small amount of tuning, we found that using $\beta_1 = 0.5$ and $\beta_2 = 0.999$ produced reasonable results.

5.1. Experiments

After training our generator, we tested it using a test set of 1000 images. To judge the quality of our model we saved the generated images alongside the noisy inputs and the original images for a qualitative view of the result. As a quantitative measure, we used the PSNR computed using the generated image and the original image. This metric is defined as

$$PSNR = 10 \log \left(\frac{MAX^2}{MSE} \right)$$

where MAX is the maximum value of a pixel (1 in this case), MSE is the mean squared error,

$$MSE = \frac{1}{C \times W \times H} \|I - I'\|_2^2$$

and I & I' are the original and generated images respectively. Since the $PSNR$ is proportional to the $\log$ of the inverse of the MSE, a higher $PSNR$ means a better quality generated image.

For the second part of our project, we fixed the generator and the weights trained for the image reconstruction task and quantized the weights using various discretization schemes. First, we tried rounding various subsets of the weights to ± 1 to see the effects of having binary weights. In addition to simple binarization we tried using the lower precision floating point implementations (single precision, half precision, etc) available. Finally, we also tried a simple linear quantization scheme in which we took the maximum and minimum of each parameter set, divided the set of intervening values into a number of allowable values (a power of 2), and rounded them to the nearest allowable value. This allowed us to simulate pseudo-fixed point precision with an easily controllable number of bits used for representation.

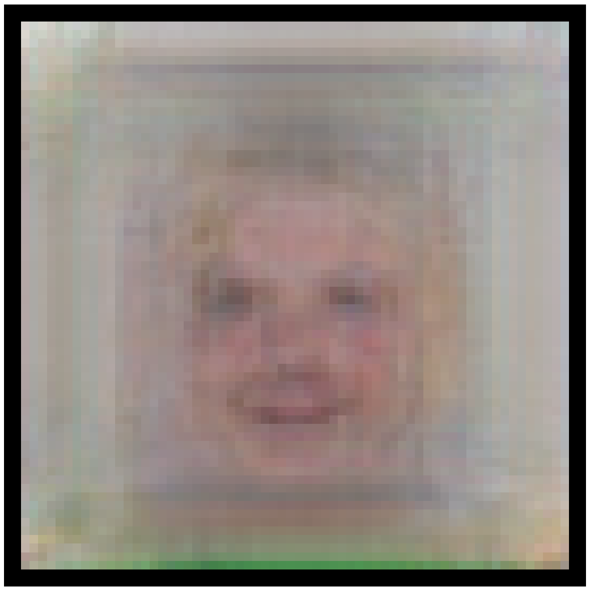


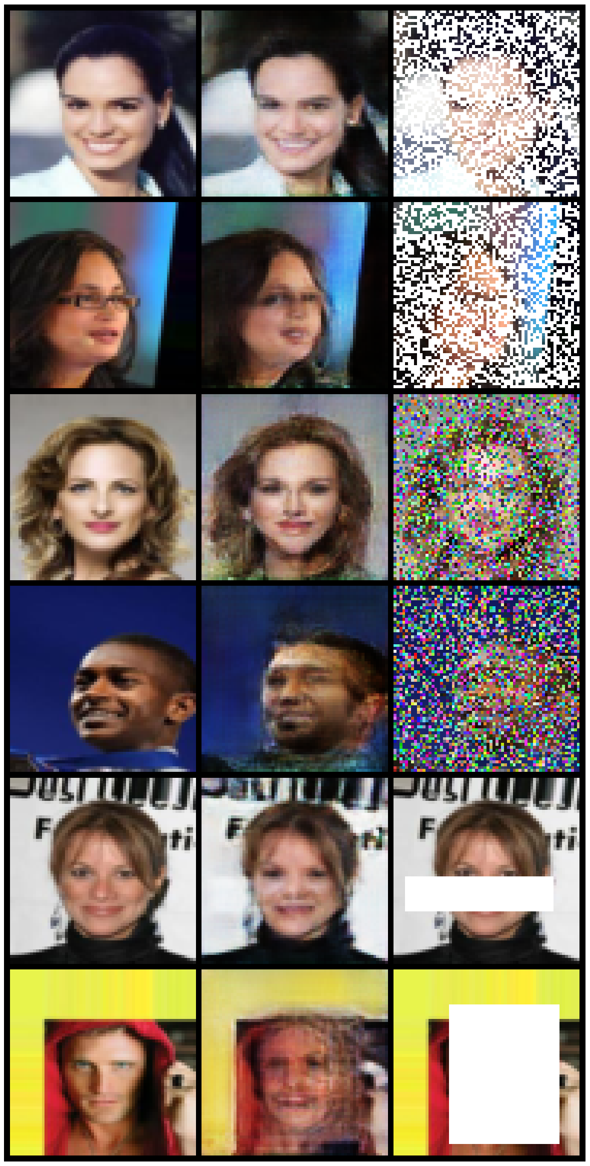
Figure 6. Output Given All White Input

5.2. Results and Discussion

Qualitative results for our method and the various quantization levels we tested can be seen in Figure 8. The left column of the image is the original image, without any noise added. The second column is after noise has been applied. The next few columns are after reconstruction using our generator with a different quantization scheme (or full precision). As can be seen from the first, third, and last rows, our generator works quite well for the discretely sampled noise types. The reconstruction is fairly realistic, although some parts are different from the original image; for example, the shine on the top row sample's forehead is muted in the reconstruction. The ease of reconstruction on these images is intuitively understandable, since the sparsity of the noise means there is usually an uncorrupted pixel close to any pixel that needs to be filled in. Similar results could probably be achieved with some kind of sophisticated interpolation.

The other type of noise, however, was far more difficult to reconstruct. In row 4 of Figure 8 we can see that removing a small strip is manageable by the generator. However in row 2, we can see that the generator produces an odd blur-like pattern in the region to be reproduced. It is even more noticeable in the row 5 sample. Additionally, the row 5 sample shows that for very large contiguous corrupted regions, the generator appears to produce a highly averaged face as a reconstruction.

The averaging of facial characteristics is easily observable if we give the generator a completely blank (white) image as input as shown in Figure 6. Lacking sufficient local context, the generator tries to output what it believes is an average face from the images it was trained on; we see



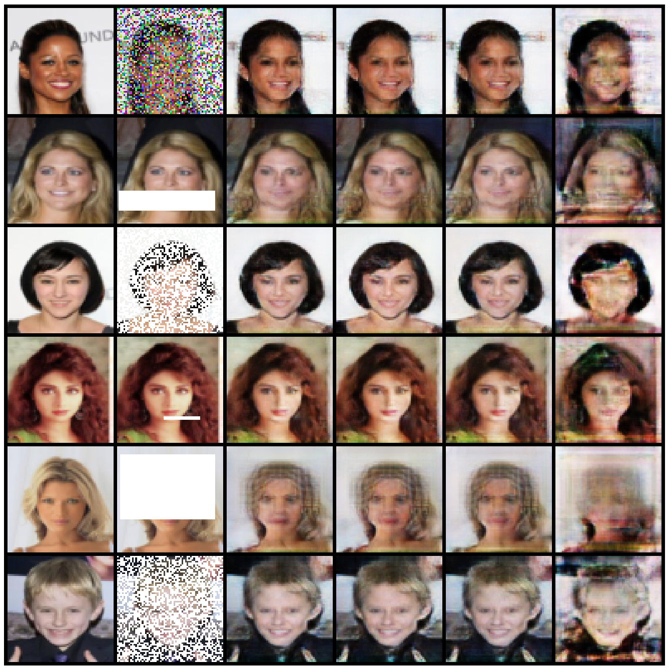
Left to right: Original, Reconstructed, Noisy
First row: Successful Reconstruction with White Noise
Second row: Failed reconstruction with White Noise
Third row: Successful Reconstruction with RGB Noise
Fourth row: Failed reconstruction with RGB Noise
Fifth row: Successful Reconstruction with White Box
Sixth row: Failed reconstruction with White Box
Figure 7. Examples of Successful and Failed Reconstructions

the resemblance of a blank face, with the bare outline of shoulders near the bottom.

In Figure 7, we show two sets of images for each type of noise: one where the reconstruction is nearly perfect, and one where it failed.

1. White Noise: Typically our reconstructions from white noise are quite good. The first row shows an example of a successful reconstruction. Most celebrities in our dataset do not wear glasses, so our model actually removes the glasses from her (row 2).

2. RGB Noise: Similarly to white noise, the model is able to reconstruct images with this noise fairly well. How-



First column: original image
Second column: corrupted image
Third column: generated image (full 32-bit weights)
Fourth column: generated image (8-bit quantized weights)
Fifth column: generated image (6-bit quantized weights)
Sixth column: generated image (4-bit quantized weights)
Figure 8. Quantization Results

	PSNR			
Model	Average	White noise	RGB noise	Box
Noise	10.3	10.3	10.3	10.3
32-bit	21.2	22.5	20.1	21.5
8-bit	21.2	22.5	20.2	21.2
6-bit	21.3	22.3	20.1	21.3
4-bit	16.1	15.6	16.3	16.5
1-bit	8.5	8.5	8.7	8.3

Figure 9. Mean PSNR for Different Quantization Results

Model	Size
32-bit	17.5 MB
8-bit	4.4 MB
6-bit	3.2 MB
4-bit	2.2 MB
1-bit	0.5 MB

Figure 10. Model size (4366595 learned parameters)

ever, you can see in our failed reconstruction, the skin color became a lot lighter and his face is blurred. This can be attributed to the imbalance of light and dark skinned celebrities in the dataset.

3. White Box: This was the toughest type of noise to handle. Unlike the other types, we cannot interpolate from nearby points. Even in our best reconstructions (row 5), there is still some blurriness. With small boxes, it is somewhat manageable, but with larger boxes, it behaves similarly to the all-white reconstruction.

The next few columns of Figure 8 show the effects of the different quantizations we tested. We see that for the most part, we can reduce the precision of the weights to half a byte before any real noticeable reduction in quality. At this point we begin to notice significant artifacts appearing across the image. Surprisingly, even reducing the weights to only 6 allowable values results in the same quality as the full precision weights. This indicates that the entire model could be compressed by around a factor of 5.3, making it much easier to store on a device with limited storage space or memory.

The quantitative results are shown in Figure 9. The first entry shows the average PSNR of completely random noise images for a batch of images from the training set. This serves as a reference point for the PSNR values. Then, we can see that all of the quantizations except the ones below 6-bit are roughly the same, around 21.2. This confirms the qualitative observation that the images produced by most of the quantizations are of similar quality. At 4-bit, however, the PSNR drops significantly which can also be seen by the drop in image quality in last column of Figure 8. At 1-bit (binary), the PSNR is even worse than the random noise and the resulting images were garbage (not shown here). Therefore, we conclude that the weight precision in generative models can be reduced significantly while achieving similar results to the full precision model, though not quite as much as in classifier models (where 1-bit binary models can be achieved, e.g. XNOR-Net as described before).

6. Conclusion and Future Work

In this paper, we have demonstrated a quantized generative adversarial network (GAN) that is capable of reconstructing images damaged by three types of noise: white noise, RGB noise, random random white rectangles. For processing 64x64 images, we achieved a model size of 3.2 MB using 6-bit quantization while achieving the same performance as the full-precision model, as measured by PSNR.

Further work would likely include trying more sophisticated architectures for the original generator/discriminator. For example, some of the existing models we reference have multi-scale discriminators and draw feature matching loss from each of them, so inclusion of multi-scale discriminators would likely help our model by allowing it to perceive both coarse and fine features in the noisy images. Switching to a patch-based discriminator could also help by allowing the discriminator to isolate specific regions it sees as fake. Additionally, since our dataset included a set of facial attribute labels we could incorporate those to allow generation of specific facial attributes.

Besides designing a more sophisticated network, we could experiment with more complex quantization techniques for model compression. This could include custom floating point formats (different number of bits and different distributions of exponent/significand fields), as well as fixed point precisions or different regularization terms during training to pull the weights towards a different set of values. Lastly, we could extend our project to video data. Since it would be unlikely for occlusion or corruption to persist in the same spots across different frames of a video, if we could extend our inputs across multiple frames it would probably be much easier to reconstruct images.

7. Contributions & Acknowledgements

Most of the work was done during group meetings. We typically met weekly for several hours at a time to work on different aspects of the project. We did do some work outside of meeting: typically Andrew tweaked architectures, Wenli helped with producing visualizations, and Alex ran experiments to tune hyperparameters.

Our code is based on PyTorch-GAN for MNIST by Erik Linder-Norn [3]. Given that the original code implements a non-convolutional GAN for the MNIST dataset, we made significant modifications to the code. These included adding layers such as convolutional layers, pooling layers, and batch/instance normalization. As well, we converted the code to generate RGB images instead of black-and-white and also changed the generator input from random noise to the corrupted image. We also had to load and preprocess our own data. In the end, we used very little from the codebase other than the general structure (e.g., training loop, model creation, etc.).

References

[1] U. Demir and G. Unal. Patch-based image inpainting with generative adversarial networks. *arXiv preprint arXiv:1803.07422*, 2018.

[2] R. Gao and K. Grauman. On-demand learning for deep image restoration. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1086–1095, 2017.

[3] E. Linder-Norn. Pytorch-gan for mnist. `https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/gan/gan.py`.

[4] L. Liu, S. Li, Y. Chen, and G. Wang. X-gans: Image reconstruction made easy for extreme cases. *arXiv preprint arXiv:1808.04432*, 2018.

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

[6] K. Nazeri, E. Ng, T. Joseph, F. Qureshi, and M. Ebrahimi. Edgeconnect: Generative image inpainting with adversarial edge learning. *arXiv preprint arXiv:1901.00212*, 2019.

[7] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.

[8] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[9] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[10] W. Tang, G. Hua, and L. Wang. How to train a compact binary neural network with high accuracy? In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[11] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[12] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[13] L. Yuan, C. Ruan, H. Hu, and D. Chen. Image inpainting based on patch-gans. *IEEE Access*, 7:46411–46421, 2019.