CT Image Generation with Generative Adversarial Networks (GANs)

Pedro Goncalves Mokarzel
Student
University of Washington, Bothell

Redmond Washington United States
pgmoka@gmail.com

1. ABSTRACT

A large field of study in machine learning is to increase the quality of CT Images during their reconstruction to eliminate artefacts. The goal of this field is to decrease the amount of radiation used in the process of CT Images. However, most machine learning algorithms require a large amount of data, which represents a problem.

The goal of this project is to use Generative Adversarial Networks to generate new CT Images from CT Image presented to the model. Work done by Bousmalis, K. has shown that Generative Adversarial Networks are able to create new images based on the input of another image [1]. The same concept will be applied here, focusing in creating CT images with low noise.

2. INTRODUCTION

Generative Adversarial Networks use two networks, a Generator and a Discriminator. The Generator receives 'original' data (different implementations change the type of data), and it creates new 'fake' data. The Discriminator's job is to determine whether one data point it receives is 'original' or 'fake' [1].

During training, the discriminator receives multiple 'original' and 'fake' images, and it decides which are 'original', and which are fake'. From that, it is possible to find the Discriminator's rate of success, and therefore, a loss. The loss can be backpropagated in the neural network for training [1].

For stability purposes, the weights of the Generator and the Discriminator Neural Networks cannot change at the same time, which means that there are two phases for one training epoch. In the first phase, only the weights of the Discriminator change. In the second phase, only the weights of the Generator change [1]. This process can be observer in Figure 2.0.

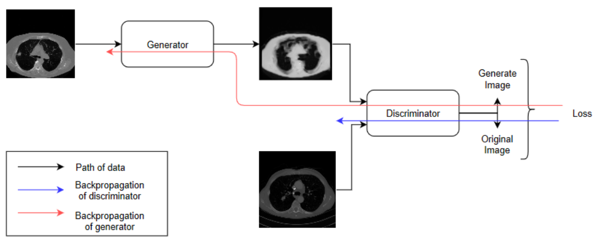


Figure 2.0: Process of GAN, showing the interactions between generator, and discriminator. Image also shows the backprojections executed during the training.

3. METHOD AND APROACH

1. The Generative Adversarial Network

The implementation of the neural networks, used for the Generator and the Discriminator, are based on multiple implementations of Generative Adversarial networks (GANs) found at the keras-GANS git repository [2]. The neural network layers are implemented with the keras package. More information on the layers, and in the project can be found at kera's original documentation [3].

The Generator:

The Generator consists of 2 sets of layers, one containing 3 subsets of layers, and the other 2 subsets of layers; all layers contain one Convolutional 2D layers (Conv2D). A visualization of these 2 sets layers can be seen in Figure 3.1.1.

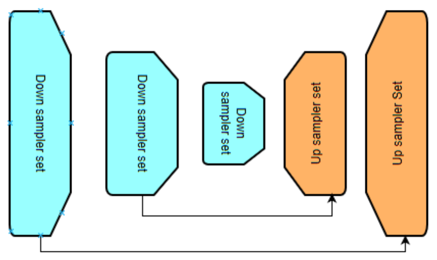


Figure 3.1.1: General view of the Generative neural network, with subsets in blue representing the 1st set of layers, and the subsets in orange representing the 2nd set of layers. The arrows show where the input of the concatenation layers from the up sampler subsets come from.

In the first set of layers of the Generator, in each subset, the Conv2D Layer is a down sampler. For all subset, except the first one, this Conv2D Layer is followed by a leaky Rectified Linear Unit (LeakyReLU) Layer, and then a Batch Normalization Layer. A visualization of this set can be seen in Figure 3.1.2.

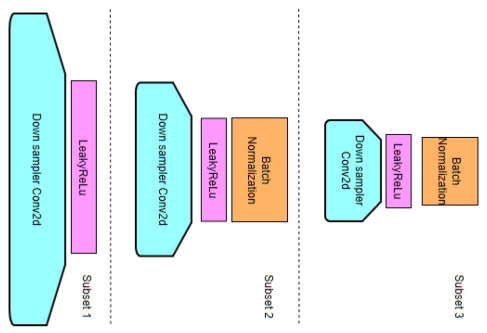


Figure 3.1.2: First set of layers of the Generator Neural Network, blue represents the down sampler Conv2d layers, pink represent the LeakyReLU layers, and orange represents the Bath Normalization layers.

In the second set of layers, the Conv2D is preceded by an UpSampling2D Layer. After the Conv2D Layer, there is a Batch Normalization Layer, and then a Concatenate Layer. The Concatenate Layer is done between the output of the current subset, and the opposite output of the previous set of layers (as seen in Figure 3.1.2). A visualization of this subset can be seen in Figure 3.1.3.

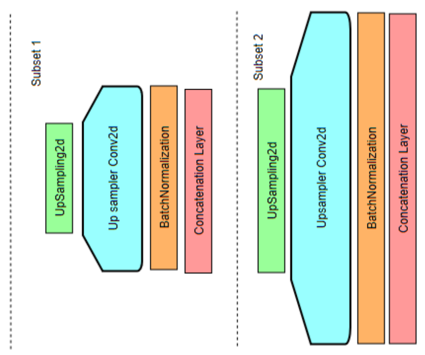


Figure 3.1.3: Second set of layers of the Generator Neural Network. Green represents the UpSampling Layer, blue represents the Upsampler Conv2d, orange represents the Batch Normalization Layer, and red represent the Concatenation Layer.

The Discriminator:

The Discriminator consists of 1 set of layers with 4 subsets of layers. All the subsets start with a Conv2D layer, and a LeakyReLU layer. All subsets, except for the first one, end with a Batch Normalization Layer (visualized in Figure 3.1.4).

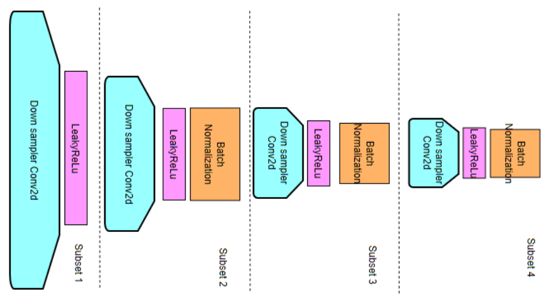


Figure 3.1.4: First set of layers of the Discriminator Neural Network, blue represents the down sampler Conv2d layers, pink represent the LeakyReLU layers, and orange represents the Bath Normalization layers.

2. Architecture

The architecture of this project is based on the keras-GAN repository [2], and the architecture used for *Comparison of Deep Learning Approaches to Low Dose CT Using Low Intensity and Sparse View Data* [4]. The project follows object-oriented architecture principles, where one class is used to create, train, validate, and save the model. For testing, a set of testing data, based on the 1-10 fold selection was pre-selected. A sample execution of the python code is outlined in a Data Flow Diagram (DFD) in Figure 3.2.2. Post analysis of the code comes from the input saved in the "Sample and Results Repository" folder.

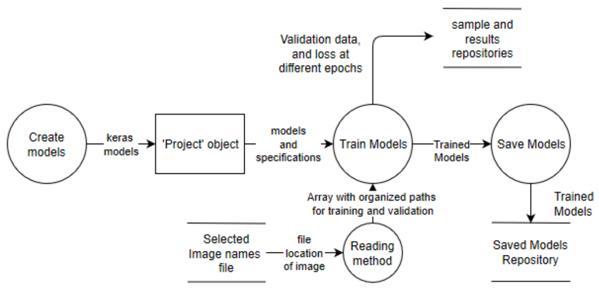
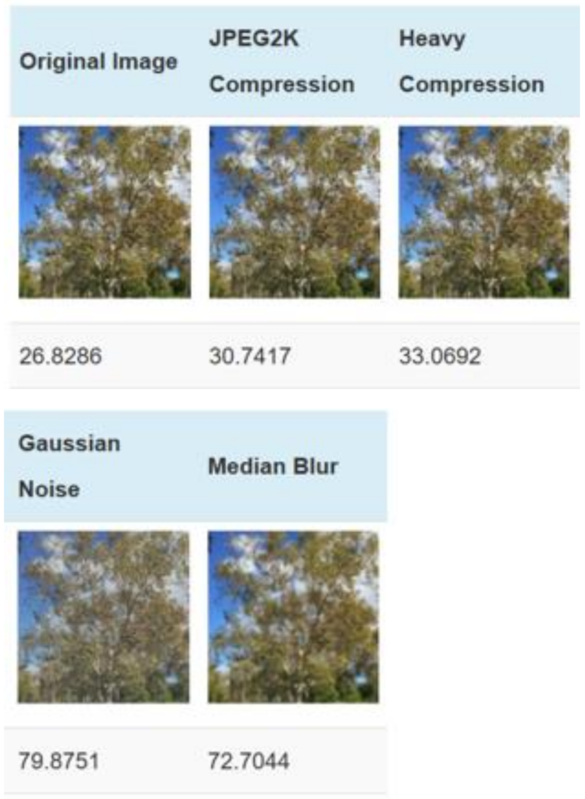

sample and results repositories
Validation data, and loss at different epochs
Create models
keras models
'Project' object
models and specifications
Train Models
Trained Models
Save Models
Array with organized paths for training and validation
Trained Models
Selected Image names file
file location of image
Reading method
Saved Models Repository

Figure 3.2.2: DFD of the execution of the model's creation, training, and saving.

3. Testing Methodology

It is important to note that CT Images, from both original and generated CT Images, are create using a method that scales the image to the minimum, and the maximum of the float point file from the CT Image. This means that CT Images(png) that are dark, with a few bright white spots, have high variance between points in the float point CT Image, and CT Images(png) that look bright have low variance between points in the float point CT Image.

The quality assessment of the images is done using a Non-Reference Image Quality Assessment method called Blind/Referenceless Image Spatial Quality Evaluator (BRISQUE) [5]. This method gives a score to an image from 0 to 100, where the smaller the score, the better the quality image. The process of BRISQUE evaluation can is done by taking the original image, extracting features, creating feature vectors from those features, and then predicting a score from those vectors. An example of BRISQUE can be seen in Figure 3.3.1.



Original Image	JPEG2K Compression	Heavy Compression
26.8286	30.7417	33.0692

Gaussian Noise	Median Blur
79.8751	72.7044

Figure 3.3.1: image shows BRISQUE score for different manipulations of an image. [6]

The BRISQUE evaluation was applied to the float point CT Image as it gives the best assessment of the image's quality. The application of BRISQUE was made in MATLAB using its native 'brisque' method [7].

Another method for assessing CT Images was Peak Signal-to-Noise Ratio (PSNR) [8]. This method is used for comparing the similarity between two images using the algorithm described in Figure 3.3.2. The larger the PSNR between two images, the more 'similar' they are.

$$MSE = \frac{\sum_{M,N} [I_1(m,n) - I_2(m,n)]^2}{M * N}$$

$$PSNR = 10 \log_{10}\left(\frac{R^2}{MSE}\right)$$

Figure 3.3.2: Image shows how PSNR is calculated. In the Image, R is the maximum fluctuation in the input image data type [8].

An example of this algorithm in action can be seen in Figure 3.3.3.

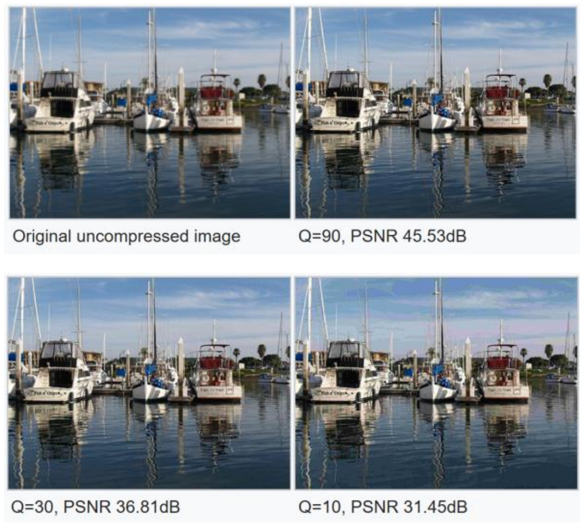
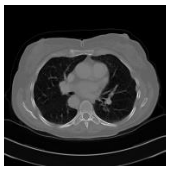
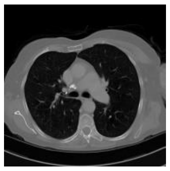
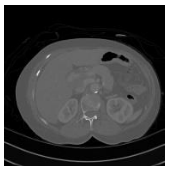


Figure 3.3.3: Image shows PSNR between an original image, and images with Q level of compression [9].

In this paper, specific comparisons will be made using the Cases seen in Figure 3.3.4.

Case 0	Case 1	Case 2
		

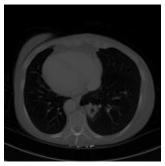
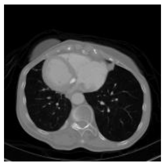
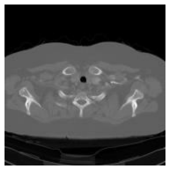
Case 3	Case 4	Case 5
		

Figure 3.3.4: Cases used for the paper. Case 0 tracks back to image 00001392_img. Case 1 tracks back to image 00003641_img . Case 2 tracks back to image 00000853_img. Case 3 tracks back to image 00003680_img , Case 4 tracks back to image 00002662_img, and Case 5 tracks back to image 00000338_img. Images gathered from *The Cancer Imaging Achieve* [10].

4. EXPERIMENT DETAILS AND RESULTS

Initial implementation aimed to check the GAN's ability to recreate a single CT Image. Figure 4.1 shows the outputs of the generator input during training.

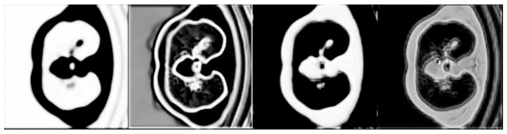


Figure 4.1: These are images generated at epoch 0, 3, 6, and 9 when using a batch of 1 for 1 image in 10 iterations, in order from left to right.

The test shows that it is possible to successfully create a CT Image from another CT Image input. This initial implementation was scaled up for the rest of the experiments by increasing the number of epochs, and the number of CT images used during training. Figure 4.2 shows outputs when a larger training data set is used.

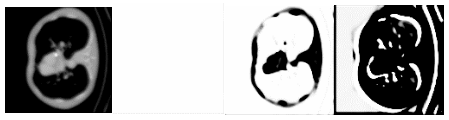


Figure 4.2: images printed at iteration 0, 3, 6, and 9 when using a batch of 2 for 12 images in 10 iterations, in order from left to right. These are images created by the Generator based on the first image of the first batch of the iteration.

From these results, it is possible to see that as the number of images increase, the number of epochs must also increase. This is due to the variance between the types of CT Images.

A factor observed during the experiments is the different implementation of ADAM optimizers. In the first implementation of the GAN, keras used its standard ADAM optimizer [3]. Following those experiments, our custom ADAM optimizer, based on work by Bousmalis, K., et all's [1] work, was implemented.

The outputs of testing data for models using 300 CT images, and 200 epochs, at different ADAM Optimizers can be seen in Table 4.3(Case 0), Table 4.4(Case 1), and Table 4.5(Case2).

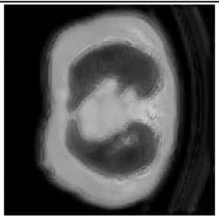
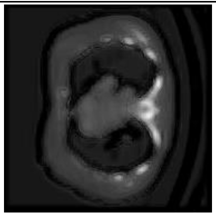
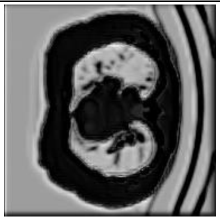
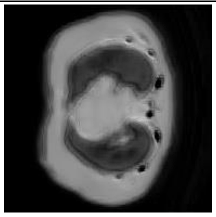
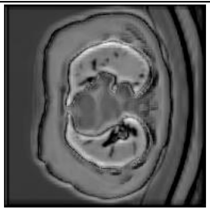
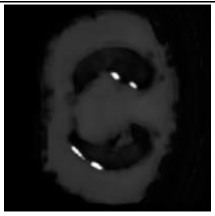
	Custom ADAM Optimizer	Default ADAM Optimizer
50 epochs		
100 epochs		
190 epochs		

Table 4.3: Shows the output at different epochs when Case 0 is the input for the generator.

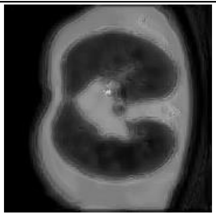
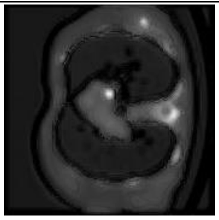
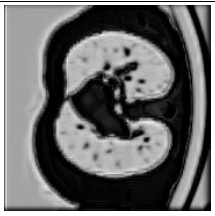
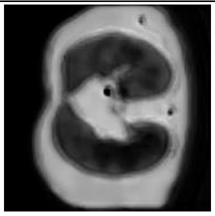
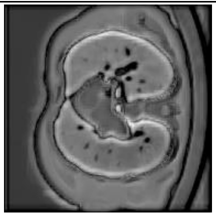
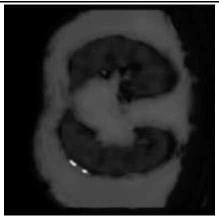
	Custom ADAM Optimizer	Default ADAM Optimizer
50 epochs		
100 epochs		
190 epochs		

Table 4.4: Shows the output at different epochs when Case 1 is the input for the generator.

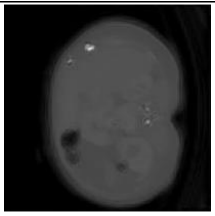
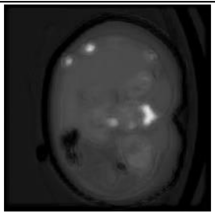
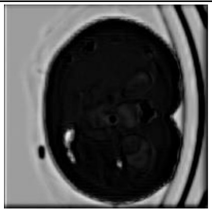
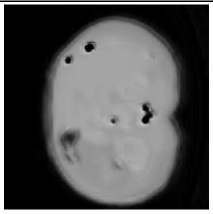
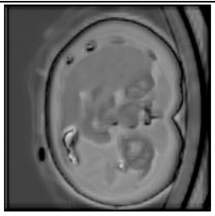
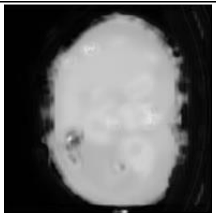
	Custom ADAM Optimizer	Default ADAM Optimizer
50 epochs		
100 epochs		
190 epochs		

Table 4.5: Shows the output at different epochs when Case 2 is the input for the generator.

From a qualitative analysis, it is possible to see that the Custom ADAM Optimizer is gaining details faster than the Default ADAM Optimizer. We can specifically look at the details gained at the 190 epochs for all Cases using the Custom ADAM Optimizer.

It is possible to see that the Custom ADAM Optimizer does a better job at keeping the float points in the correct range, as its png images are much closer to the range of an original CT Image. We can specifically look at Case 0 (Table 4.4), and Case 1 (Table 4.5) to see how extreme the float points range are when using the Default ADAM Optimizer.

One problem the Custom ADAM Optimizer has is that during training it did not learn to properly fully separate the person from the background, as seen from all Cases. We can see specifically in Case 0 (Table 4.4) at 190 epochs, when using the Custom ADAM Optimizer. In the image, it is possible to see that the background is uniform, until it approaches the person. In the transition between the background, and the person, we see an increase on the floats. The transition ultimately was able to create a hard edge, standing out against the Default ADAM Optimizer.

Another factor analyzed was the BRISQUE score of images generated. It is possible to see the progress of the model's generated CT Images at different epochs for Case 0, Case 1, and Case 2 on Figure 4.5, Figure 4.6, and Figure 4.7

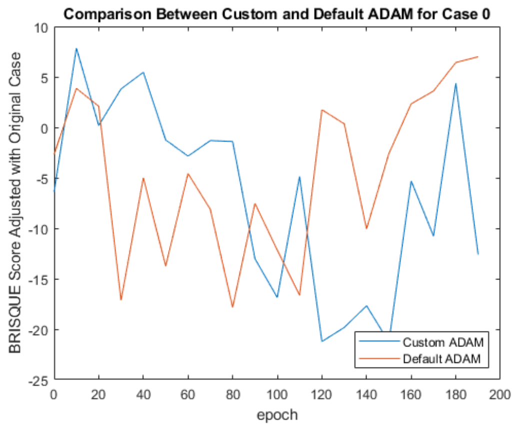


Figure 4.5: BRISQUE Score of CT Images generated from Case 0 at different epochs. Score has been adjusted to the BRISQUE score of Case 0 (35.2146).

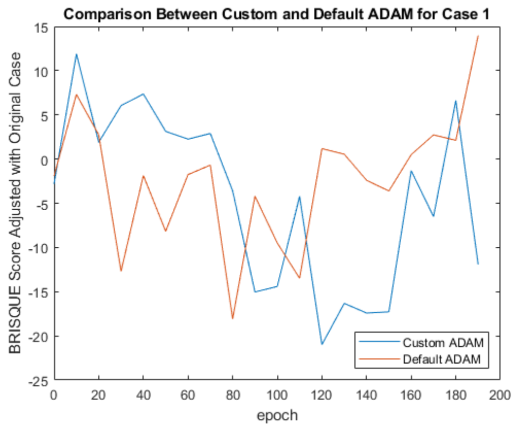


Figure 4.6: BRISQUE Score of CT Images generated from Case 1 at different epochs. Score has been adjusted to the BRISQUE score of Case 1 (38.3816).

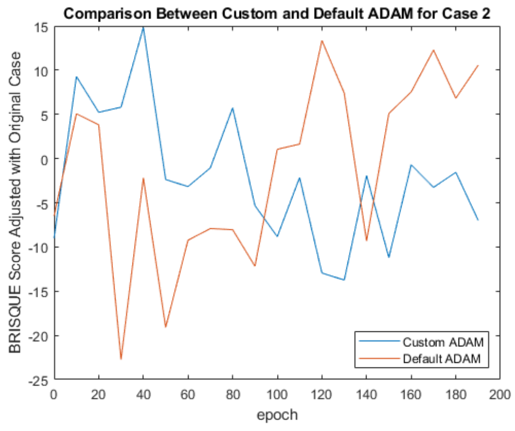


Figure 4.7: BRISQUE Score of CT Images generated from Case 2 at different epochs. Score has been adjusted to the BRISQUE score of Case 2 (36.3730).

It is possible to see that for all Cases, the final epoch image for the model using the Custom Adam Optimizer had a smaller BRISQUE score than the Default Adam Optimizer.

The model will generate the negative of the CT Image in certain initial iterations when using the Custom Adam Optimizer. This may be the reason why the model has a problem in properly creating the background.

The calculated mean of BRISQUE of a sample of 354 CT Images was 36.9365. When applying BRISQUE to the generated CT Images, the average BRISQUE score of all images generated through epochs by the model using the Custom ADAM Optimizer is smaller than the average BRISQUE score of all images generated by model using the Default ADAM Optimizer. This can be seen in Table 4.8.

	Default Adam Optimizer	Custom Adam Optimizer
Case 0	33.8434	31.6291
Case 1	32.2042	30.7261
Case 2	35.2446	34.2042

Table 4.8: Average of CT Images generated between epochs 0 and 200 for Case 0, Case 1, and Case 2.

This shows that, as the two models train, the Custom Adam Optimizer model creates images with higher quality than the Default Adam Optimizer. The BRISQUE score for all averages are also lower than the original BRISQUE score of their original CT Image.

The expectations are that, as further epochs proceed, The BRISQUE score will continue to oscillate, eventually staying between the average BRISQUE score of all CT Images used for the model's training, and the BRISQUE score of the model's input

It is important to note that having the lowest BRISQUE score does not mean the Generator is creating the best CT Image. We see when looking at The Custom Adam Optimizer using model for Case 0 that at epoch 160, the image has the lowest BRISQUE score. This is due to the fact that details found on the original Case 0 decrease the BRISQUE score. We can see this in detail in Figure 4.9.

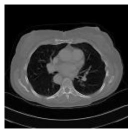
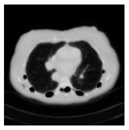
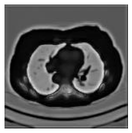
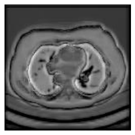
Original CT Image	20 Epoch Generated CT Image	160 Epoch Generated CT Image	190 Epoch Generated CT Image
			
38.3816	38.2039	17.0963	25.7844

Figure 4.9: Images with their associated BRISQUE score.

Figure 4.9 Shows that the original CT Image has the highest BRISQUE score. We see a curve from the BRISQUE scores because the generator goes through the following steps:

1) The model just makes a blurred copy of the input, creating an image with a very similar BRISQUE score to the input's BRISQUE score.
2) The model gets the general shape first, without any specific details. The BRISQUE score decreases as the shape becomes smooth.
3) The model gets mode details, such as veins, and bones. The BRISQUE score to increases as the smoothness of the image is lost with details from the original CT Image.

The custom Adam optimizer was chosen to create a new model, a model using 3000 CT images for training. Case 3, Case 4, and Case 5 are used to analyze the model. It is possible to see the differences for the outputs of models trained with 300 vs 3000 CT Images in Table 4.10.

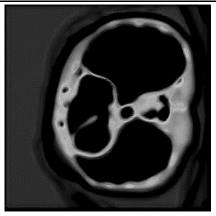
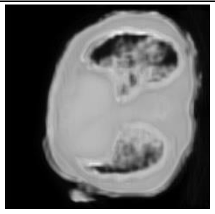
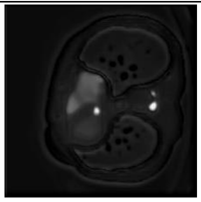
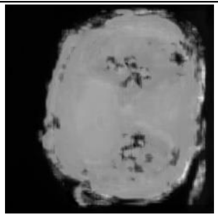
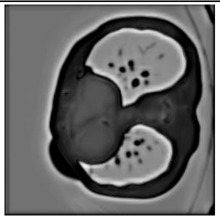
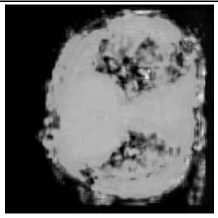
	Output Using 300 CT Images for Training	Output Using 3000 CT Images for Training
60 epochs		
120 epochs		
180 epochs		

Table 4.10: Shows output difference between model trained with 300 CT Images against the output of a model trained with 3000 CT Images for Case 4.

From Table 4.10, we can see that the new model (model using 3000 images for training) struggled to gain details for the 'inside' of the CT Image. It also struggled to make the full separation between the person and the background. It is possible to see the error between the person and the background are similar to the error seen when using the Standard Adam Optimizer model trained with 300 CT Images to create CT Images. It is possible to see that those borders become more solid between epoch 120 and epoch 180 of the new model.

The model trained with 3000 images never flipped to its negative during the different epochs, contrasting to the model trained using 300 images. This might be due to that 'flipping' happening inside of epochs, as the number 'flops' conducted between each epoch was 10 times large on the new model. It may also be due that the higher number and variety of images helped the model identify the background more properly.

The model trained with 3000 images was able to achieve a much higher level of quality for printing veins inside of the lungs.

Outputs for Case 3 and Case 5 can be seen in Figure 4.11 and Figure 4.12.

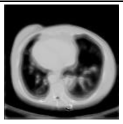
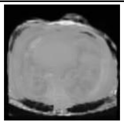
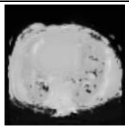
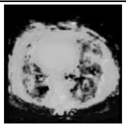
0 Epoch Generated CT Image	60 Epoch Generated CT Image	120 Epoch Generated CT Image	180 Epoch Generated CT Image
			

Figure 4.11: Output at different epochs for model trained with 3000 images for Case 3.

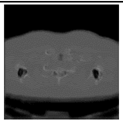
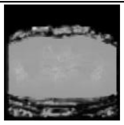
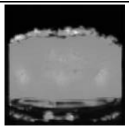
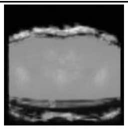
0 Epoch Generated CT Image	60 Epoch Generated CT Image	120 Epoch Generated CT Image	180 Epoch Generated CT Image
			

Figure 4.12: Output at different epochs for model trained with 3000 images for Case 5.

Case 3 outputs are similar to outputs of Case 4, as expected, due to the similarities between Case 3 and Case 4. Case 5 has the least amount of visible detail. This might be due higher amount of variance between the 'filled' part between CT images. The BRISQUE score of the created CT Images when using the new model can be seen in Figure 4.13.

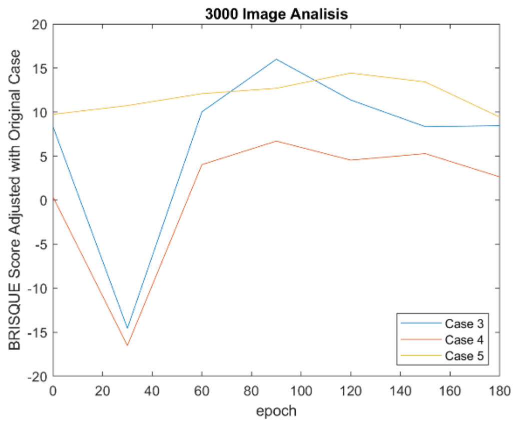


Figure 4.13: BRISQUE scores for Case 4, Case 5, and Case 6 outputs at different epochs when the model trained with 3000 images was used. Each Case's output was adjusted to the BRISQUE score of its original image.

It is possible to see for that for the new model, for outputs from Case 4, Case 5, Case 6, the pattern observed in the older model, for outputs from Case 1, Case 2, and Case 3, can be seen. For the image, there is a drop in the BRISQUE score, followed with a jump up, and then an approach to 0. This shows that, while the outputs for the two models are significantly different, they both follow the same pattern.

PSNR was used to test the inputs and outputs of Case 3, and Case 4 to check if the new model is properly creating new CT Images. Those comparisons can be seen in Figure 4.14, and Figure 4.15.

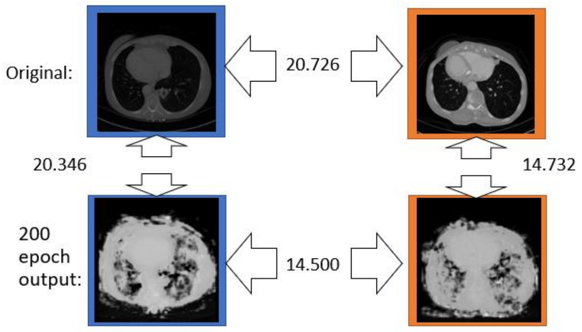


Figure 4.14: Image shows PSNR comparison between the Original Case 3 image (blue image on top left), Original Case 4 image (orange image top right), output after 200 epoch's for Case 3 image (blue image on bottom left), and Case 4 image (orange image on bottom right). Color background associates original CT Image to its output after the generator.

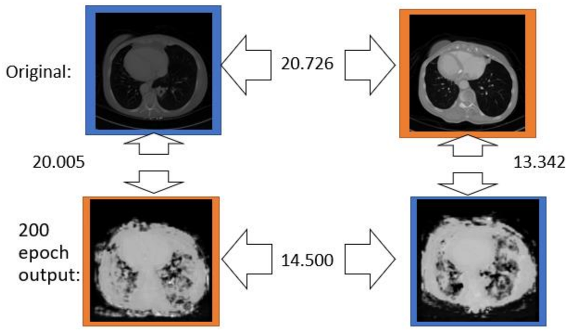


Figure 4.15: Image shows between PSNR comparison between the Original Case 3 image (blue image on top left), Original Case 4 image (orange image top right), output after 200 epoch's for Case 3 image (blue image on bottom right), and Case 4 image (orange image on bottom left). Color background associates original CT Image to its output after the generator.

These two cases were selected due to their similarity. The goal is to see patterns in the Generator's output. It is possible to see by looking at the PSNR scores the creativity of the Generator. The points taken from the test are that:

1) The PSNR between Case 3, and Case 4 was larger than the PSNR between the 200 epoch output for Case 3, and Case 4. (from Figure 4.14). From Point 1, it is possible to see that the generator is

'being creative' in its process of creating each new CT images. This shows that the Generator is not (1) applying a static filter to its input, and that (2) the Generator is not making a copy of its starting input.

2) The PSNRs for Case 3 and Case 4 are higher with their associated generated CT Image than the PSNRs between them, and their non-associated generate CT Image. (from Figure 4.14 and Figure 4.15). From this, it is possible to see that the images created by the Generator share aspects with their original CT Image, and that CT Images created from other CT Images are not more similar than CT Images created from their original CT Image. This shows that our generator's input is making a difference in the process of creating parameters. If Point 6 was not true, we would be able to see that the input is not having a significant impact on the output.

3) From Point 1 and Point 2, we can see that the Generator is taking an input and a creating new CT Image based on it. We also see that this new CT Image is different to other created CT Images as the generator is.

4) The PSNR value between Case 3 and the 200 epoch output for Case 3 is close, but smaller, than the PSNR value between Case 3 and Case 4. (from Figure 4.14). From this, it is possible to see that created CT Images are not too closely related to the input that created them.

5) The PSNR value between Case 3 and the 200 epoch output for Case 4 is close, but smaller, than the PSNR value between Case 3 and Case 4. (from Figure 4.15). From this, it is possible to see that other created CT Images do not have the same similarity than the input that created them.

6) From Point 4 and Point 5, it is possible to see that CT Images created from the Generator stand out against original CT Images, independent of their relationship input.

7) The PSNR value between Case 4 and the 200 epoch output for Case 4 is close, but lager, than the PSNR value between Case 4 and the 200 epoch output for Case 3. (from Figure 4.14). From this, it is possible to see that CT Images created from the Generator stand out from each other from the perspective an original CT Image.

8) The PSNR value between Case 4 and the 200 epoch output for Case 3 is close, but smaller, than the PSNR value between the 200 epoch output for Case 4 and the 200 epoch output for Case 3. (from Figure 4.15). From this, it is possible to see that the relationship between CT Images created from the Generator is closer than the relationship between a created CT Image, and an unassociated original CT Image.

9) From Point 7 and Point 8, we that, while a created CT image has a closer relationship to its original CT Image than to another created CT Image, another created CT Image has a closer relationship to another created CT Image than their input. This suggests that trends being applied by the Generator are reflected in its created CT Images.

5. CONCLUSION AND DISCUSSION

The goal of the research in this paper was to develop a model that is able to create new CT Images that looks like other original CT Images using an Artificial Neural Network based on the work done in *Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Network* [1]. While this paper laid the path for achieving this with the current Artificial Neural Network architecture, the CT Images created during the time frame of the delivery were not satisfactory to the parameters set in the abstract.

When looking at different potential optimizers to be used in the creation of CT Images, the Custom Adam Optimizer creates CT Images that have better image, are closer to the expected float range, and have more detail. It is possible to conclude that the Custom Adam Optimizer will converge faster to a point where the output of the Generator is undistinguishable from a possible original CT Image.

When testing models created with 200 epochs and 300 CT Images for training, the CT Images generated had a higher amount of detail in the 'inner' parts of an individual, and a better edge between the person and the background, but the difference between its background and the person was not satisfactory. When testing models created with 200 epochs and 3000 CT Images for training, the CT Images did a better job a creating 'empty' spaces such as the background and lungs, but it struggled to generate the details seen inside a person. The new model also struggled to create proper edge between the person and the background. This is due to the variance between the different CT Images, causing details to be harder to gain. It is possible to conclude that, to create a proper CT Image with the amount of data as is, in the range of 200 epochs, the optimal number of CT Images for training is between 300 and 3000. It is also possible to conclude that if the variance between images was smaller, the desired CT Images would be reached faster. Therefore, if data was preprocessed, such that CT images trained on had low variance between each other, Generator outputs would be closer to desired results, but possible CT Images created would have a smaller range of possible outputs.

By looking at the BRISQUE scores, it is possible to see the adjusted BRISQUE scores CT Images created by the Generator will approach 0 as the number of epochs increase. From this, it is possible to conclude that, as long as the

Generator and the Discriminator remain stable, the quality of the created CT Images is going to become closer to the quality of a normal CT Image as the number of epochs increases. Therefore, if the model trained with 3000 images is continued to be trained, the quality of its output will continue to increase.

By looking at the PSNR analysis, it is possible to conclude that the GAN is in the track to achieve the goals of the paper. This is due to how Point 1 to Point 8 in the PSNR analysis demonstrate how CT Images created are different from other created CT Images, the input makes a difference in the process of creating CT Images, created CT Images are different from other original CT Images, a created CT Image is different from its input CT image, and created CT Images interactions with original CT Images match interactions between original CT Images and other original CT Images. Point 9 is a natural factor of GANs, but it is important it is kept track of, as if it increases, the GAN may start taking it images and using it to create 1 type of CT Image.

Another metric to check results from this paper is look at outputs from the model found at Pix2Pix GitHub [2], whose code is based on the same base paper of this paper [1] . The main differences between this paper and Pix2Pix is their Artificial Neural Networks, and data used as Generator input. This paper's Generator implemented 1 entry convolutional layer, 2 down-sampling layers, and 2 up-sampling layers. The Pix2Pix's Generator's implemented 1 entry convolutional layer, 6 down-sampling layers, and 6 up-sampling layers. Some sample outputs from different models using Pix2Pix can be seen in Figure 5.0.

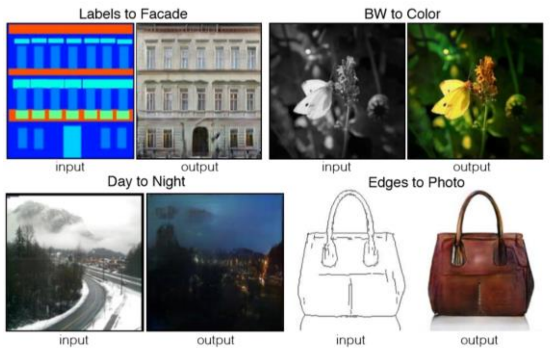


Figure 5.0: Outputs from model found in the Pix2Pix GitHub. [2]

It is possible to see that images from Pix2Pix models have a higher quality than the model presented in this paper. This might be due to the necessity of more epochs to train the model, the more complex Artificial Neural Network implemented by Pix2Pix, or the type data used for training. From the comparison, it is possible to conclude that a more complex Artificial Neural Network may improve the quality of outputs.

Another analysis can be made with *A Style-Based Generator Architecture for Generative Adversarial Networks* [11]. This paper implemented a different architecture with GANs to create new faces. Some of their results can be seen in Figure 5.1.

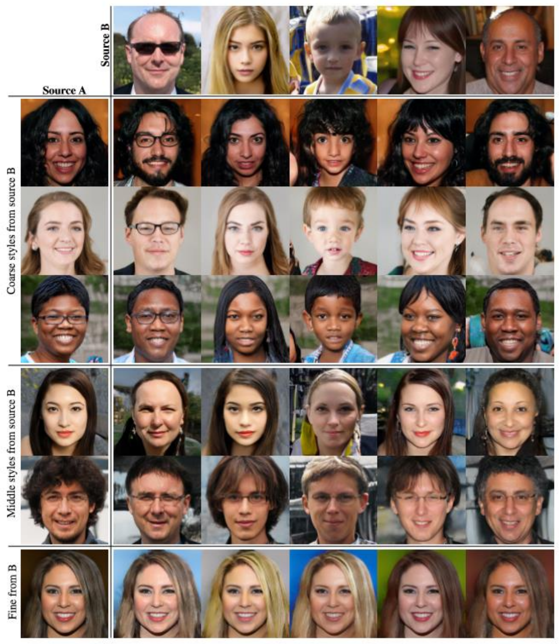


Figure 5.1: As described in the paper, "Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions (42 – 82) brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes,hair,lighting)andfinerfacialfeaturesresembleA.Ifweinsteadcopythe styles of middle resolutions(162 – 322) from B, weinherit smaller scale facial features, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved. Finally, copying the fine styles (642 – 10242) from B brings mainly the color scheme and microstructure"[11].

It is possible to see that these images are able to combine faces and create new ones. When comparing the results from the Artificial Neural Network Implemented on this paper, and the one implemented in *A Style-Based Generator Architecture for Generative Adversarial Networks*, it is possible to see that their Generator outputs have a qualitative higher image quality than the Generator outputs from this paper (compared to each owns own data). The one problem from *A Style-Based Generator Architecture for Generative Adversarial Networks* is that it aims to combine 2 images, which decreases overall variety in the outputs. It is possible to conclude that, while the model implemented in this paper might not currently have the same quality in images, it has a higher variety in its outputs.

6. FUTURE WORK

As outlined in the conclusion, the image outputs will continue to increase as the number of epochs increase. Leaving the model to train with the current set of CT Images may eventually yield the desired results.

Currently, for 3000 images for 200 epochs, training time takes approximately 2 weeks. Processes for selecting data that have less variance between each other can create a smaller sample size that will yield better results. For initial large-scale training, this path might be recommended so that a model that creates satisfactory outputs is created faster. That would allow for a larger number of models to be created in a smaller spam of time, allowing further analysis of what paths to take.

While PSNR was a good metric to analyze the bit differences between, another interesting metric for measuring similarities between CT Images is Structural Similarity Loss (SSIM Loss).

A more complex Artificial Neural Network, such as the one implemented in Pix2Pix [2], may increase the quality of the model. However, as seen in the conclusion, a different Artificial Neural Network implemented in *A Style-Based Generator Architecture for Generative Adversarial Networks*[11] has created human faces with high quality. Therefore, in future work, the architecture laid out in *A Style-Based Generator Architecture for Generative Adversarial Networks*[11] could be explored to increase the quality of the images. One thing to be kept in mind is that the variety with the new architecture may decrease due to how it creates new image.

Another set of Artificial Neural Networks used to create new data is Variational Auto Encoder (VAE). Creating CT Images proved to be a non-trivial task for Generative Adversarial Networks due to the amount of details in the image. CT Images could be an interesting parameter to measure the efficiency and quality of images generated by VAEs vs the efficiency and quality of images generated by GANs. The comparison between the networks could be achieved with metrics set on this paper (and SSIM Loss).

7. ACKNOWLEDGEMENTS

Work done on this paper was completed during the author's enrollment in the class CSS 490, *Introduction to Machine Learning*, offered in the University of Washington, Bothell, taught by Professor Dong Si.

The author presented results every week during meetings involving other CT Imaging related research overseen by. Professor Thomas Humphries, and Professor Dong Si.

8. REFERENCES

[1] Bousmalis, K., et all. 2017. Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial Network. *2017 IEEE Conference on Computer Vision and Patter Recognition* (CVPR) 95-104.
[2] Linder-Noren, E. 2017. Keras-GAN. *GitHub*. https://github.com/eriklindernoren/Keras-GAN
[3] Chollet, F., et all. 2018. Keras: The Python Deep Learning Library, Keras Documentation.
[4] Humphries, T., Si, D.Coulter, S., Simms, M., Xing, R. 1 March 2019. Comparison of deep learning approaches to low dose CT using low intensity and sparse view data. *Medical Imaging 2019: Physics of Medical Imaging*. DOI 10.1117.
[5] Mittal, A., Moorthy, A., & Bovik, A. 2012. No-Reference Image Quality Assessment in the Spatial Domain. *IEEE Transactions on Image Processing, 21*(12), 4695-4708.
[6] Shrimali, K. R.. Image Quality Assessment: BRISQUE. June 3, 2018. *Learn OpenCV*. Retrieved from: www.learnopencv.com/image-quality-assessment-brisque/
[7] MathWorks. 2019. Brisque. *Documentation*. Retrieved from www.mathworks.com/help/images/ref/brisque.html.
[8] MathWorks. PSNR. 2019. *Documentation* . Retrieved from. www.mathworks.com/help/vision/ref/psnr.html
[9] Peak Signal-to-noise Ratio. (n.d.). Retrieved from en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.
[10] Clark, k. et all. *The Cancer Imaging Archieve(TCIA): Maintaining and Operating a Public Information Repository*, Journal of Digital Imaging, Volume 26, Number 6, December, 2013, pp 1045-1057.
[11] Karras, T., Laine, S., & Aila, T. (2018). A Style-Based Generator Architecture for Generative Adversarial Networks.