Week 5 Project: TomoGAN

Seth Howells  
seththowells@lewisu.edu  
DATA-56600-002, Spring I 2022  
Digital Image ProcessingLewis University

Sara Tahan  
sarastahan@lewisu.edu  
DATA-56600-002, Spring I 2022  
Digital Image ProcessingLewis University

# Introduction

Generative Adversarial Networks (GANs) are a type of deep-learning neural network used in unsupervised machine learning and for digital image processing [1]. Like the convolutional neural network (CNN), GANs use multiple convolution layers while removing noise from images. GANs, however, are built from a system using two neural networks, the Generator and Discriminator, that work off one another to analyze and process the images within the dataset. The Discriminator functions much like the CNN with multiple hidden layers and a purpose of image classification. The Generator is quite the opposite and is an Inverse Convolution Neural Net. It is a generative algorithm that, based off the input data, uses its ‘imagination’ to produce its own image.

The Generator’s purpose is to produce fake image samples to trick the Discriminator in its ability to distinguish between a real image and a fake image. They compete with one another while this process is repeated multiple times until they both have improved at their tasks. The Generator seeks to find the probability that the Discriminator will make a mistake, while the Discriminator works to determine the probability that the image it has received is from the actual training data rather than the Generator [1]. This process helps the Discriminator to train more efficiently and minimize loss and will continue until the Discriminator is no longer identifying fake images. The Discriminator also functions to help train the Generator to denoise the images.

Although there are a few different variations of the GAN model, the focus of this paper will be on a TomoGAN model. TomoGAN is a GAN-based method that is designed specifically to improve the quality of high-resolution images [2]. TomoGAN is a highly effective denoising technique that works well for poor image conditions. To build a TomoGAN model, there are multiple convolution layers separated by layers of 2D Max Pooling, 2D Upsampling, and Concatenation. The convolution layers remove high-dimensional features from the input images by transforming it. In these layers, a kernel smaller than the actual input image is passed over the image where the average of the neighboring pixels to the center pixel is used to replace the center pixel [3], producing a feature map. The 2D Max Pooling layers compute the maximum value for the patches in the feature map and use that to output a downsampled, or pooled, feature map. This method reduces the size of the images by reducing the pixel count in the output of the previous convolution layer. Max Pooling is used to allow the neural network to look at larger portions of the image at once and to help reduce overfitting in the model [4]. The 2D Upsampling layers are used in the Generator and double the dimensions of the input image. They are simple, unweighted layers that typically follow a convolution layer in GAN models [5]. Upsampling is required in GAN models to generate an output image, and although it does no learning itself, the convolution layer that follows interprets its output. Concatenation layers link together inputs of the same size to stack them into a 1xN vector image [1]. This is done to help maintain and increase the processing speed of the neural network and typically follow a layer of Upsampling.

In the TomoGAN architecture, the loss (1) for the Discriminator is determined by,



where *D* is the Discriminator and *G* is the Generator, and the Generator loss is determined by the weighted average of the Adversarial loss (2), the Perceptual loss (3), and the Pixel-wise mean-squared error (MSE) (4).

2

3

4

While the MSE is commonly used to determine loss in image processing, it can also contribute to blurred output images, something that is often noticed in CNN models. In the GAN and TomoGAN methods, the Generator’s loss uses the MSE along with two other loss functions to generate the false images as well as have a similar MSE to the ground truth image [2].

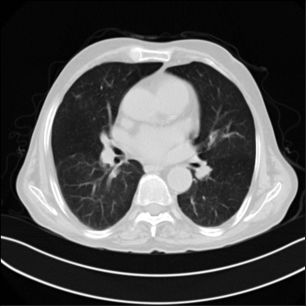
The remaining sections of this report will discuss the data, methodology, results, and analysis. Section II provides detailed information regarding the images used in this analysis. Section III describes the methodology of processing the images, while Section IV reports the results. Finally, Section V will provide an analysis of the results.

# Data Description

A dataset with 100 images of malignant tumor scans was used for this analysis. The selection of images was downloaded from a larger dataset of images with normal, malignant, and benign scans [6]. The original images varied in pixel size but were all adjusted to 1024 x 1024 for this analysis. Noise was added prior to running the TomoGAN model and was done so at random using a built noise function in Python. Figure 1 below is an example image of a malignant tumor scan with no noise added (a ground truth image) and Figure 2 shows the same image after noise has been added.

1. Image Data

| **Image Class** | **Original Image Size** | **Adjusted Image Size** | **Object Type** | **Description** |
| --- | --- | --- | --- | --- |
| Clean | 512 x 512 px | 1024 x 1024 px | Malignant Tumor | Scan of a malignant tumor |
| Noisy | 512 x 512 px | 1024 x 1024 px | Malignant Tumor | Scan of a malignant tumor with noise added |



1. Ground truth image of malignant tumor.

A black and white image of a person's face

Description automatically generated with low confidence

1. Same image with noise added.

# Methodology

Image denoising using the TomoGAN method was performed using Python on Google Colab Python (version 3.6.9) [7]. The images from the dataset were split into two folders: ‘Train’ and ‘Test.’ The ‘Train’ folder had a total of 80 images and the ‘Test’ folder a total of 20 images. The image directory was imported into Google Colab, keeping the training and testing images separate.

Diagram

Description automatically generated

1. Flow chart of the TomoGAN model [2].

Once the images were preprocessed, noise was added at random. Next, the pre-trained TomoGAN model was imported [8] to run on the images. A pre-trained model was used due to the amount of time required to train on GPU clusters. Figure 3 above shows a flow chart of the TomoGAN model. The architecture of the TomoGAN model involved 15 hidden convolution layers and can be seen at the end of this report.

# Results

Table II at the end of this report shows the model summary of the TomoGAN analysis after it was complete. The convolution layers filtered the images to work to remove noise. The layer widths for the convolution layers are as follows: three convolutions at 8, 32, and 32 followed by a pooling layer, then two convolutions at 64 each followed by a pooling layer, then two convolution layers at 128 each followed by the final pooling layer, then one convolution layer at 128 followed by Upsampling and Concatenation, then two convolutions at 64 each followed by Upsampling and Concatenation, then two convolutions at 32 each followed by the final Upsampling and Concatenation layers, finally ending with four convolutions at 32, 32, 16, and 1 as the output layer.

A picture containing text

Description automatically generated

1. Noisy (left), clean (center), and denoised (right) images after running the TomoGAN model.

A picture containing set

Description automatically generated

1. Noisy (left), clean (center), and denoised (right) images after running the TomoGAN model.

Figures 4 and 5 above show the results of the TomoGAN model run on the images of malignant tumor scans. In both figures, the noisy image can be seen on the right, the ground truth image in the center, and the denoised output on the left. Adjustments were made to improve the results of the denoised image. For example, the images were converted to grayscale, and it was ensured that the image size of all inputs was changed to 1024 x 1024. The image range was also adjusted from 200 – -100 to 90 – -30 to get a better view of the output. However, neither output image is as clear as the ground truth images.

# Analysis

Due to how long it takes to run this digital image processing model, a pre-trained TomoGAN model was used. The original Python code for the TomoGAN model is from a research paper done on X-ray tomography image restoration [2, 8]. It was expected that by plugging in the new image dataset, the results would be like the results of the research paper in that the denoised images would be closer to the ground truth image than the noisy image. In theory, the denoised images from the TomoGAN model analysis should yield cleaner images than those from the CNN model analysis. Here, the denoised images from the TomoGAN model are not very clean. Although some of the noise has been removed, there is still some present and the colors of the image have been inverted (Figures 4 and 5 above). The denoised images are also blurred, making it difficult to distinguish some of the detail of the images. While the exact cause of the denoised images’ subpar results is unknown to these authors, there are multiple reasons a TomoGAN denoising model may yield inadequate results.

The first possible issue is something called mode collapse. Mode collapse is an issue with the Generator model in that it has trouble generating different fake images, and instead generates multiples of the same fake image [9]. In this instance, there will be little diversity among the generated images. Having repeated generated images sent to the Discriminator model hinders its ability to train properly. This can produce inadequate output images and increased loss values.

Another possibility is convergence failure. This is one of the more common ways a GAN model can have issues. In convergence failure, the Generator and Discriminator that should be working in tandem, fail to find a balance with one another [9]. It is typically caused by the Generator creating images that are too easy for the Discriminator to identify as fake, resulting in a loss value that is close to zero for the Discriminator, and one that is quite high for the Generator. The denoised images that are produced often still have noise or static present, which make it difficult to discern the actual image.

Vanishing gradient can cause the model to train slower than it normally would. This occurs when the Discriminator is *too* good and does not feed enough relevant information into the Generator for it to properly learn and train [3]. Here, the derivatives become smaller and smaller due to the weights inability to change their values. Using a pre-trained model from a successful research paper, the authors assume that the issue is not necessarily with the model itself. The way the images are formatted prior to running through the TomoGAN model can have a great impact on the results. If the images were not formatted correctly, whether their size, color, or preprocessing was poorly formatted, the resulting denoised images could have less-than-ideal results. Although the authors were unable to definitively determine that this was the cause, it is still a possibility that formatting is to blame.

##### References

1. “Generative Adversarial Networks (GANs): An introduction,” *GeeksforGeeks*, 06-Mar-2019. [Online]. Available: https://www.geeksforgeeks.org/generative-adversarial-networks-gans-an-introduction/?ref=lbp. [Accessed: 10-Feb-2022].
2. Z. Liu, et al, “Tomogan: Low-dose synchrotron X-ray tomography with Generative Adversarial Networks,” *Journal of the Optical Society of America A*, vol. 37, no. 3, pp. 422–434, Jan. 2020.
3. S. Barua, S.M. Erfani, and J. Bailey, “FCC-GAN: A fully connected and convolutional net architecture for GANs,” *arXiv e-prints*, vol. arXiv-1905, May 2019.
4. “Max pooling in convolutional neural networks explained,” *DeepLizard*, 16-Feb-2018. [Online]. Available: https://deeplizard.com/learn/video/ZjM\_XQa5s6s. [Accessed: 12-Feb-2022].
5. J. Brownlee, “How to use the upsampling2D and conv2D layers in Keras,” *Machine Learning Mastery,* 12-Jul-2019. [Online]. Available: https://machinelearningmastery.com/upsampling-and-transpose-convolution-layers-for-generative-adversarial-networks/. [Accessed: 12-Feb-2022].
6. A. Karmakar, “Tumour\_Multiclass\_Dataset,” *Kaggle*, 04-Feb-2021. [Online]. Available: https://www.kaggle.com/amitkarmakar41/tumour-multiclass-dataset. [Accessed: 11-Feb-2022].
7. “Welcome to Colaboratory,” *Google Colab*. [Online]. Available: https://colab.research.google.com/. [Accessed: 08-Feb-2022].
8. “AIScience Tutorial: Denoising TomoGAN,” *GitHub*. [Online]. Available: https://github.com/AIScienceTutorial/Denoising. [Accessed: 08-Feb-2022].
9. J. Brownlee, “How to identify and diagnose GAN failure modes,” *Machine Learning Mastery*, 20-Jan-2021. [Online]. Available: https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/. [Accessed: 12-Feb-2022].
10. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

TomoGAN Model

1. Model Summary

| **Layer** | **Type** | **Output Shape** | **Param #** |
| --- | --- | --- | --- |
| INPUT\_1 | Input Layer | (None, None, None, 1) | 0 |
| CONV2D | Convultion2D | (None, None, None, 8) | 16 |
| CONV2D\_1 | Convolution2D | (None, None, None, 32) | 2,336 |
| CONV2D\_2 | Convolution2D | (None, None, None, 32) | 9,248 |
| MAX\_POOLING2D | Max Pooling2D | (None, None, None, 32) | 0 |
| CONV2D\_3 | Convolution2D | (None, None, None, 64) | 18,496 |
| CONV2D\_4 | Convolution2D | (None, None, None, 64) | 36,928 |
| MAX\_POOLING2D\_1 | Max Pooling2D | (None, None, None, 64) | 0 |
| CONV2D\_5 | Convolution2D | (None, None, None, 128) | 73,856 |
| CONV2D\_6 | Convolution2D | (None, None, None, 128) | 147,584 |
| MAX\_POOLING2D\_2 | Max Pooling2D | (None, None, None, 128) | 0 |
| CONV2D\_7 | Convolution2D | (None, None, None, 128) | 147,584 |
| UP\_SAMPLING2D | UpSampling2D | (None, None, None, 128) | 0 |
| CONCATENATE | Concatenate | (None, None, None, 256) | 0 |
| CONV2D\_8 | Convolution2D | (None, None, None, 64) | 147,520 |
| CONV2D\_9 | Convolution2D | (None, None, None, 64) | 36,928 |
| UP\_SAMPLING2D\_1 | UpSampling2D | (None, None, None, 64) | 0 |
| CONCATENATE\_1 | Concatenate | (None, None, None, 128) | 0 |
| CONV2D\_10 | Convolution2D | (None, None, None, 32) | 36,896 |
| CONV2D\_11 | Convolution@d | (None, None, None, 32) | 9,248 |
| UP\_SAMPLING2D\_2 | UpSampling2D | (None, None, None, 32) | 0 |
| CONCATENATE\_2 | Concatenate | (None, None, None, 64) | 0 |
| CONV2D\_12 | Convolution2D | (None, None, None, 32) | 18,464 |
| CONV2D\_13 | Convolution2D | (None, None, None, 32) | 9,248 |
| CONV2D\_14 | Convolution2D | (None, None, None, 16) | 528 |
| CONV2D\_15 | Convolution2D | (None, None, None, 1) | 17 |
| Total Parameters: | 694,897 | | |
| Trainable Parameters: | 694,897 | | |
| Non-trainable Parameters: | 0 | | |