CT Image Generation with Generative Adversarial Networks (GANs)

Pedro Goncalves Mokarzel
Student
University of Washington, Bothell

Redmond Washington United States
pgmoka2@gmail.com

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.

KEYWORDS

CT Image: Image created from an X-ray. Done from the reconstruction of information got from the X-ray.

Noise: interference that obscure image features.

Artifacts: interference that appears to be an image feature.

1 The Generative Adversarial Network

1.1 A Brief Introduction to GANs

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].

1.2 The Project's Neural Networks

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]. (boogie woogie)

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 1.2.1.

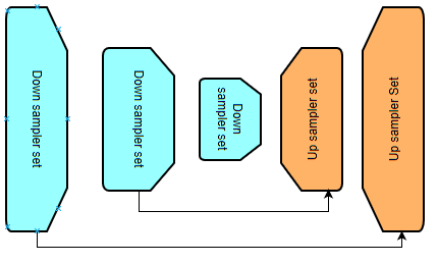

Figure 1.2.1: General view of the Generative neural network, with subsets in blue representing the 1st set of layers, and the subsets in orange representing the 2nd set of layers. The arrows show where the input of the concatenation layers from the up sampler subsets come from.

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 1.2.2.

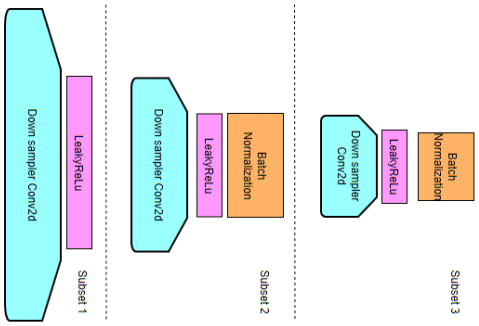

Figure 1.2.2: First set of layers of the Generator Neural Network, blue represents the down sampler Conv2d layers, pink represent the LeakyReLU layers, and orange represents the Bath Normalization layers.

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 1.2.1). A visualization of this subset can be seen in Figure 1.2.3.

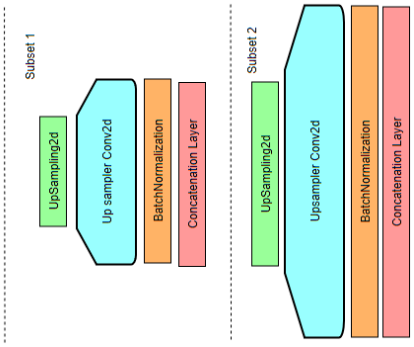

Figure 1.2.3: Second set of layers of the Generator Neural Network. Green represents the UpSampling Layer, blue represents the Upsampler Conv2d, orange represents the Batch Normalization Layer, and red represent the Concatenation Layer.

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 1.2.4).

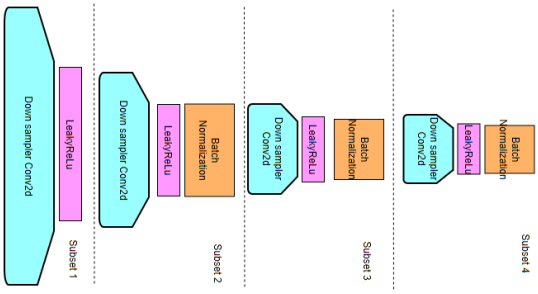

Figure 1.2.4: First set of layers of the Discriminator Neural Network, blue represents the down sampler Conv2d layers, pink represent the LeakyReLU layers, and orange represents the Bath Normalization layers.

2 The Architecture and Procedure

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. A simple class diagram is outlined in figure 2.1.

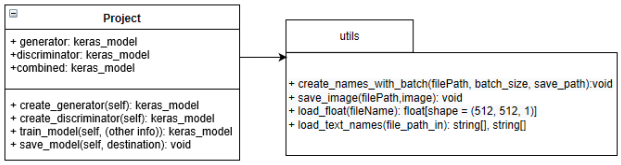

Figure 2.1: Current simple class diagram of project.

In the current implementation, four file repositories are necessary. A 'model' repository to save the models, a 'data' repository for input, a 'results' repository for saving outputs, and a 'sample' repository for saving images that are created while training.

The division of data was done by the 'create_names_with_batch' method. The output of this method was written to a test file with the path for each image. The text was read by the 'train_model' method to properly select images for training.

For tests, user passed arguments are used to change batch number, and epochs easily. This information is passed to the class at the object's creation which creates the Generator and Discriminator models. The 'train_model' method trains the models, receiving information on specifying epochs, input repositories, and output repositories. A sample execution is outlined in a Data Flow Diagram (DFD) in Figure 2.2.

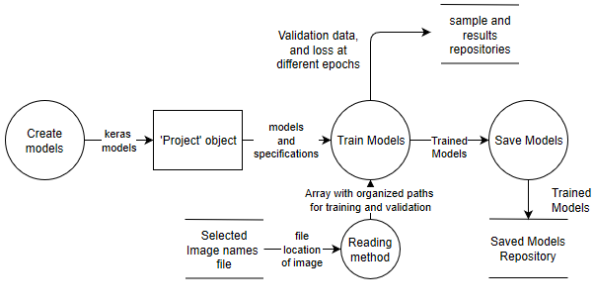

Figure 2.2: DFD of the execution of the model's creation, training, and saving.

3 Data

The data available for training is 3954 CT Images from the QIN LUNG CT dataset [5]. This data has already been preprocessed for training and saved in float format (for training) and PNG format (for analysis) from Professor Humphries CT Imaging research [4].

The whole dataset was downloaded to a server at the University of Washington (DAIS 1) for training. During development phase, a sample of 300 images was downloaded locally for development. Small samples, up to 30, were selected for training in the local machine.

4 Current Results

Two important tests have been done. The first is the training in one CT Image. This shows that the model is able to generate CT images (results in figure 4.1).

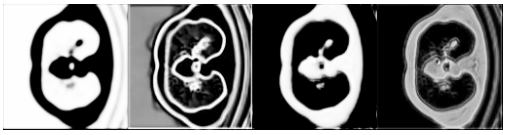

Figure 4.1: These are images generated at iteration 0, 3, 6, and 9 when using a batch of 1 for 1 image in 10 iterations, in order from left to right.

The second was the test with batch implementation in data with larger variance. This showed that batch helps the training of images with high variance, and that it is possible to achieve results while training with multiple CT images (results in figure 4.2).

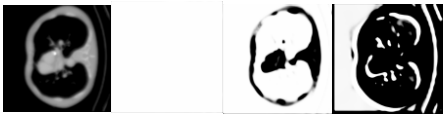

Figure 4.2: images printed at iteration 0, 3, 6, and 9 when using a batch of 2 for 12 images in 10 iterations, in order from left to right. These are images created by the Generator based on the first image of the first batch of the iteration.

5 Achieved Goals

As of November 4th, 2019, a version of this project for small data samples has been created. Models created from this have shown that it is possible to generate new CT images.

Batch implementation has been implemented with a file system that optimizes memory usage. Batch implementation has helped in processing images with high variance between images

A GitHub repository with code from this project has been created. It aims to keep version control, and distribution of findings [6].

6 Next Steps

A few steps are necessary before large scale training begins. These are:

1. Create methods for the for validating model.
2. Implement validation while testing.
3. Implement the saving system for models.
4. Migrate to DAIS 1.
5. Use of GPU for keras training.
6. Determine success standards.

After these steps, different data quantities will be trained with different epochs will be able to be tested and created in DAIS 1. Those shall be in multiples of 3 (batch size for final report). Those results shall be reported in the final report.

The final report shall display those results, and outline what model generates the better image. Guidelines for the use of the project shall be kept at its GitHub [6].

7 Potential Goals

One of the potential implementations of this project is to use data outputs to train reconstruction algorithms. The LEARN algorithm [1] has already been used in the reconstruction of the same data used for this project. It would be interesting to see the outcomes of LEARN when trained on created images from this project.

In initial tests, there are interesting results when different batch numbers and epochs are used. A comparison of the different results between these different models could be inciteful.

Another goal would be to determine quantitative ways to determine the success of the model. This could be done using a PSNR comparison to the original image and the generated image to determine 'creativity', but a method for determining the quality of the image needs to be found.

REFERENCES

[1] Bousmalis, K., et all. 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. Keras-GAN. 2017. *GitHub*. https://github.com/eriklindernoren/Keras-GAN

[3] Chollet, F., et all. *Keras: The Python Deep Learning Library,* Keras Documentation*, 2018.*

[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] 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.

[6] Goncalves Mokarzel, P. GAN for CT Image Generation. *GitHub*. https://github.com/pgmoka/GAN-for-CT-Image-generation