Using Generative Adversarial Networks to Produce Synthetic Overhead Imagery

Adalid Helguero, Twinkle Gera, Bawer Alissa, Amir Itayem, and Mohammad Saad

ahelguer@gmu.edu, tgera@gmu.edu, balissa@gmu.edu, aitayem@gmu.edu, msaad4@gmu.edu

George Mason University

Lockheed Martin

Abstract -

Currently, the process of collecting images with satellites is an expensive process for a trivial return.. The purpose of this project is to create images similar to those that satellites create so that the process of collecting these images can be automated and be much cheaper. To solve this problem and produce cheap satellite imagery, Generative Adversarial Networks were developed**.**

Two adversarial networks will be built into a Deep Convolutional Generative Adversarial Network (DCGAN): the generator and the discriminator networks. The generator network will produce synthetic images, while the discriminator network will determine the accuracy of those images. These two networks will work together to improve the images that the generator network produces by making them look as realistic as possible. These synthetic images will provide Lockheed Martin with a dataset for their future project(s). Not only that, but they can be used for other training purposes at a lower cost than obtaining real satellite images from a more expensive approach.The stakeholders of this project are customer and subject matter expert: Tim Parker and Johnathan Brant.

Progressive GANs are a GAN training process that allows for stable training and is capable of outputting high quality images. The training involves starting with tiny images and adding more layers that increase the generator and discriminator model until the image size is achieved.

Key Terms - Machine Learning, DCGAN, Keras, Tensorflow, Generator, Discriminator

I. **Introduction**

A. Overview -

The requirements of this project include developing a generator neural network component that is capable of creating realistic synthetic overhead imagery. During this process the discriminator will create components to improve the accuracy of the images produced by the generator. The discriminator will be responsible for validating the authenticity of those synthetic images. This will allow the generator to produce realistic satellite images based on the discriminators response.

Together, these components will learn to produce the desired fabricated imagery. The DCGAN was designed by avoiding fully connected layers and implementing others, such as convolutional and pooling. These layers make up the generator and discriminator using the Keras deep learning library, and Tensorflow as the backend.

Using a mini batch gradient descent, the generator model will be trained by the discriminator. A stable DCGAN will have a loss curve for both the discriminator and generator, which will start high and progressively decrease over each epoch. At the end of each epoch, the program will print the loss of both models onto the console. The purpose of this is to monitor the loss and terminate the process if the training becomes unstable. The method of validating the DCGAN and the Progressive GAN is qualitative. The users will determine if they look realistic enough. The end goal is to be able to produce fabricated images with stable GANs that are similar to the testing dataset.

It is important to note that most synthetic images produced by GANs are based on simplistic ones from small to medium size dimensions. However, the data set that was used in this project had large dimensions. The sizes that the images were chipped, varied from 8x8 pixels up to 256 x 256 pixels while the originals are around 2900 x 3100 pixels. These images are taken from the xView dataset, which contains a vast collection of them caught by satellites with large dimensions. These satellite images have many different types of objects with higher resolutions, while this could be beneficial for the accuracy of machine learning, having a huge dataset will also cut down the training rate drastically. For this reason, a chipping function was utilized to downsample the images into smaller sizes.

B. Background

The application of the DCGAN and Progressive GAN is image generation. The main difference between the two generator models is how the architecture is designed. The designed DCGAN works only with a specific image size, while the progressive GAN will build up to the final dimensions. For example, the progressive GAN will start with an 8x8 image and then will increase the size by building another layer on top of the previous layer. It will pass down the previous model's weight, until the desired image size of 256x256 is achieved.

II. **Implementation**

A. TensorFlow

TensorFlow is an open source mathematics library that can be applied to machine learning for neural networks. TensorFlow was used for its libraries to provide the backend support for Keras.

B. Keras

Keras is an open source neural network Python library that can be run on top of TensorFlow, Theano, or PlaidML.This library was used to design the layers in the generator and discriminator. The built in functions hold parameters for the training process. Examples of this include filter, strides, and padding, which affect the tensor dimensions once a convolutional layer takes place.

C. Convolutional Layer

One of the most common methods in image processing for machine learning is convolutional neural networks. These neural networks classify images into tensors or arrays. The tensors are then convolved with a filter, which will produce a new tensor. These tensors will have deeper dimensions from the channels, but the width and length will shrink down.

The idea for convoluting is to find patterns from all adjacent pixels. Since all adjacent pixels are correlated, patterns are always present. This produces the weights related to the spatial area. These weights are then refined once more as training takes place. This is all built into the deep learning environment, Keras. The Keras framework allows for more intuitive convolutional layering with built in functions.

D. Generator

The generator neural network component is modeled and trained to create realistic synthetic imagery. The generator is combined with the discriminator as they go back and forth trying to learn how to make the images realistic as possible.

First, the generator will take a vector that contains random noise data for images. This random noise vector goes through the generator and creates the fake images. The first layer will add a fully connected layer called “dense” into the generator architecture, which is an important step to set the weights in the neural network. The convolutional layers will then be added until the final tensor has the dimensions of 256x256.

E. Discriminator

The discriminator is built with the convolutional 2D, dropout, flatten, and dense layers. During training, the fake images created by the generator are combined with the real images from the dataset. The discriminator determines whether these images are authentic or synthetic. These results are provided back to the generator, which uses this data to improve the synthetic imagery to make them more realistic, until it can deceive the discriminator. The generator is penalized if it produces implausible results. The discriminator will take an image and apply convolutional layers until one dimension noise vector is left. It will then use the vector to predict if the image is synthetic or real.

F. Training/Hyperparameters

During the training of the DCGAN, the generator consists entirely of convolutional layers with the addition of pooling layers. The absence of fully connected layers was a fundamental part of this DCGAN[1]. The random noise vector discussed before has a size of 100 random samples that was put into the generator that contains the images of the xView dataset. The size of each of these images were 256x256, but they are upsampled from 8x8. The activation layer used is LeakyReLU. This is the Leaky version of the rectified linear unit function, allows a small gradient when the unit is not active. BatchNormalization is used after each convolutional layer is called. This normalizes the activations of the previous layer after each batch[3].

The discriminator takes in a 256x256 image from the generator and down samples it by half until the size is 8x8. Similar layers from the generator are used again in this discriminator.

For the progressive GAN training, layers are added based on the loss from each model. Once the loss for the generator and discriminator are flattening or plateauing, another layer is added to deepen the architecture. The training starts with a single convolutional layer and a 8x8 pixel size that gets deeper over time.

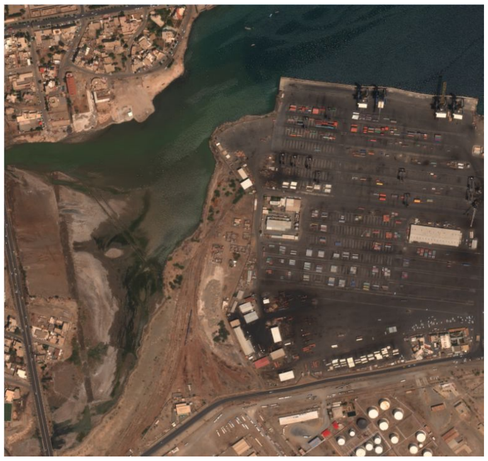
After the first generator and discriminator are built, the weights are constructed and saved. The second models will be created with the dimensional size of 16x16 pixels. The weights are then loaded into the models from the previous training done by the 8x8 models. This is repeated until the 256x256 dimensions are met.

In order to find when to add a layer during which epoch, the loss was closely monitored. After multiple epochs of training, a new layer was added at the average epoch value that the loss was plateauing.

The training method implemented contains inputting the random noise vector, initializing the variable for fake and real images, and defining the hyperparameters. The entire dataset used is approximately 113,000 images which are cut up or chipped pieces from the xView dataset. The batch size is 100 images, which are randomly selected from the entire 113,000 images each time. The epoch size varies, but averages around 9 batches.

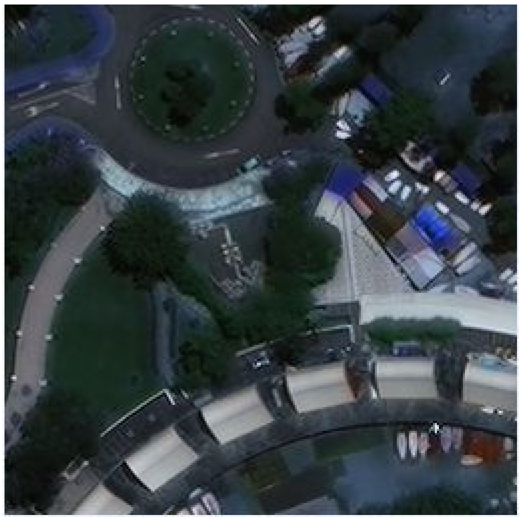
To find the average epoch in which the loss starts to flatten, training was done from start to finish multiple times. For the DCGAN, having a batch size of 200 or more was ineffective and did not create the images desired. A batch size of 100 or 150 produced the highest quality images and optimized the training time for both GANs.

G. Figures and Tables -



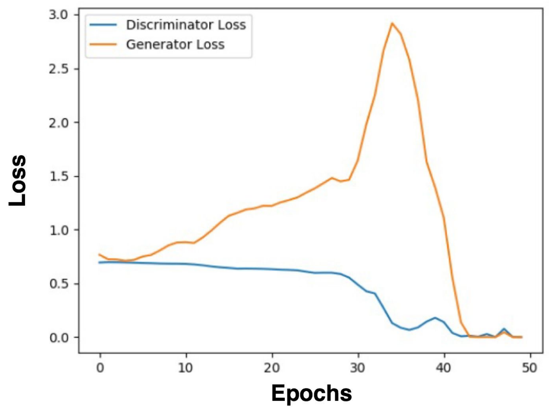
[Figure 1]

The image above (Figure 1) is part of the xView dataset. These images are around 3000x2900 in size and are satellite images of cities. They contain buildings, vehicles, trees, and roads. These images were too large and would have increased our training time substantially.



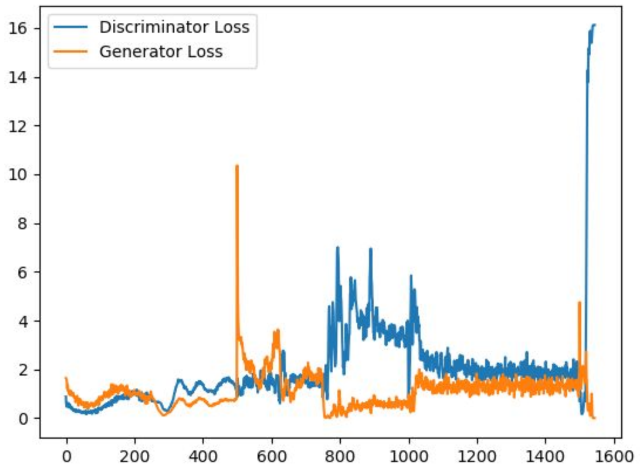
[Figure 2]

Figure 2 is a smaller image that was chipped from the larger ones from the xView dataset. This chip is an example of a 256x256 image. These were used in training instead of the originals.



[Figure 3]

Figure 3 is a graph that depicts the loss over each epoch for both the generator and discriminator. This graph depicts the loss for the DCGAN. Binary cross-entropy is the loss function that the models are following. This graph shows the loss over each epoch for 50 epochs. During this specific run, the loss was very low.



[Figure 4]

Figure 4 is another loss graph that depicts the loss of the progressive GAN. The progressive GAN was trained for 60 epochs. It is following the same loss function as the DCGAN, but for a longer amount of training time. This specific run had a high loss near the end where the images that were generating were 256x256 in pixel size. This was graphed differently from the DCGAN. This is graphed in a way where the loss for each image that is genereated is plotted.

H. Future Work

GANs have many applications. An extension of this could be object detection and classification. The models for each GAN can also learn how to detect objects discussed before; roads, buildings, vehicles, and trees. This additional application can further advance the use of GANs by adding another layer of sophistication. These extra labels can help generate even more complex imagery.

III. **Conclusion**

GANs can be utilized to produce synthetic images. This is done by providing a training set of images that the program uses to train itself therefore creating realistic images. The generator produces images which the discriminator tries to classify as either real or fake.

DCGANs and Progressive GANs are used to create the synthetic images. The DCGAN was created by removing fully connected layers and implementing convolutional and pooling layers. When the DCGAN is stable it will have a loss curve for the discriminator and generator, which starts high and decreases over each epoch. The Progressive GAN, which is a DCGAN built differently, will build up to the final dimensions. During the training, the layers of the Progressive GAN are added based on the losses from each model. When the losses start flattening, another layer is added to deepen the architecture. The training started with one convolutional layer and 8x8 pixel size then gets deeper over time.

In conclusion, GANs are powerful neural networks that use a generator and discriminator neural network, which competes with each other. The generator tries to "trick" the discriminator by trying to pass synthetic images off as realistic. The discriminator network's goal is to catch any fake images that the generator produces. The final product from the GAN is synthetic imagery, which the discriminator network believed to be real based on the data provided from the xView dataset. The images produced by the GANs from this project is to make satellite imagery more accessible and cheaper for others.

References

https://www.tensorflow.org/guide/keras
Keras.io
[1]https://towardsdatascience.com/deeper-into-dcgans-2556dbd0baac
[2]https://keras.io/layers/advanced-activations/
[3]https://keras.io/layers/normalization/
[4]https://machinelearningmastery.com/introduction-to-progressive-growing-generative-adversarial-networks/