**Using a Deep Convolutional Generative Adversarial Network to Produce Synthetic Overhead Imagery**

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

[ahelguer@gmu.edu](mailto:ahelguer@gmu.edu), [tgera@gmu.edu](mailto:tgera@gmu.edu), [balissa@gmu.edu](mailto:balissa@gmu.edu), [aitayem@gmu.edu](mailto:aitayem@gmu.edu), [msaad4@gmu.edu](mailto:msaad4@gmu.edu)

George Mason University

Lockheed Martin

***Abstract -***

The purpose of this project is to improve the process of tagging data to be used for the training of machine learning object detection and classification models. This object detection process can be used in the context of tagging data including the ability to tag keywords to organize data efficiently.

Two adversarial networks will be built into a Deep Convolutional Generative Adversarial Network (DCGAN): the Generator and the discriminator. The Generator Network will produce synthetic images and the discriminator Network will determine the accuracy of those images. A training model will automate this process at the end of the DCGAN architecture. These synthetic images will provide users with authentic details in classified areas to gain knowledge or for future programs. After the DCGAN Network goes through the process, it will generate which images are real and which ones are fake for the user. The stakeholders of this project are our customer and subject matter expert: Tim Parker and Johnathan Brant.

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

I. **Introduction**

1. Overview -

The requirements of this project include creating a Generator Neural Network component that is capable of generating 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. Together these components will produce realistic synthetic overhead imagery. The DCGAN was designed by removing fully connected layers and implementing other layers, such as convolutional, core, 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 to learn how to create synthetic overhead imagery from satellites. 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 system 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 is qualitative since the images produced by it will determine if they look realistic enough. The end goal is to be able to produce synthetic images with a stable DCGAN that are similar to the testing dataset.

It is important to note that most synthetic images produced by GANs are based on simplistic images from small to medium size dimensions; however, the data set that was used in this project have large images. These images are taken from a xView dataset, which contains a vast collection of images taken 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.

1. Background

DCGAN and Progressive GANs

are used to create a reliable Machine Learning training set. This will include tagged synthetic overhead images. GANs that produce synthetic overhead images are suitable for training a neural network, which will perform object detection and classification. The implementation of a deep convolutional generative adversarial network versus a progressives generative adversarial network for the model has been used to see which provided less noise in the synthetic images. The main difference between the two generator models are the dimensional sizes of the images. While the DCGAN works with the dimensions of the desired image size, the progrossive GAN will build into the final dimensions. For example, the progressive GAN will start by a 4x4x3 image and then progressively will increase by building another “build\_generator” or “build\_discriminator” model on top of smaller models, until the desire image size is achieved.

II. **Implementation**

1. Tensorflow

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

1. Keras

Keras is an open source Neural Network Python Library that can be run on tensorflow, R, Theano, or PlaidML. This project implemented keras with tensorflow as the backend. This library was used to code the layers in the generator and discriminator. These built in functions hold parameters for the imagery processing. Examples of this include filter, strides, and padding, which affect the tensor dimensions once a convolutional layer takes place.

1. Convolutional Layer

One of the most common methods in image processing for machine learning are 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 with smaller dimensions with one dimension that would be wider. This is a convolutional layer.

The idea behind this is to produce a one dimensional array using multiple convolutional layers. The computer can then analyze the array using the sigmoid function and classify if the array comes from an authentic image or not. This is all built into the Deep Learning Environment, Keras. Keras then uses Tensorflow as the backend. The Keras framework allows for more intuitive convolutional layering with built in functions.

1. Generator

A generator neural network component is created to generate realistic synthetic overhead imagery to create plausible data. The neural network is then expanded to include objects of interest, geospatially tagged objects, and objects classified on the synthetic overhead images. The generator is integrated into the GAN, essentially it will learn from the discriminator using the numpy arrays.

First the generator will build images using the architecture, which is designed in the ‘build\_generator’ function. It will start from a 100 dimensional noise vector. 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. Then the transposed convolutional layers will be added until the final tensor has the dimensions of the desired image size in pixels which is 256x256x3. The dimensions consist of the length and width of the image and the last dimension shows if it's in color or grayed scale.

1. Discriminator

This discriminator is built with the convolutional 2D, dropout, flatten, and dense layers. The discriminator inspects the images produced by the generator and determines whether 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 then penalized by the discriminator if it produces implausible results. The discriminator will take an image and apply convolutional layers until a one dimension noise vector is left. The discriminator will then use the vector to predict if the image is synthetic or real.

1. Training/Hyperparameters

During the training of the DCGAN, the generator consisted entirely of convolutional layers with the addition of pooling layers. The absence of fully connected layers was a fundamental part of this DCGAN[1]. A random noise vector with a size of 100 random samples was put into the generator along with a numpy array containing the image data 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, which 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, which normalizes the activations of the previous layer after each batch[3].

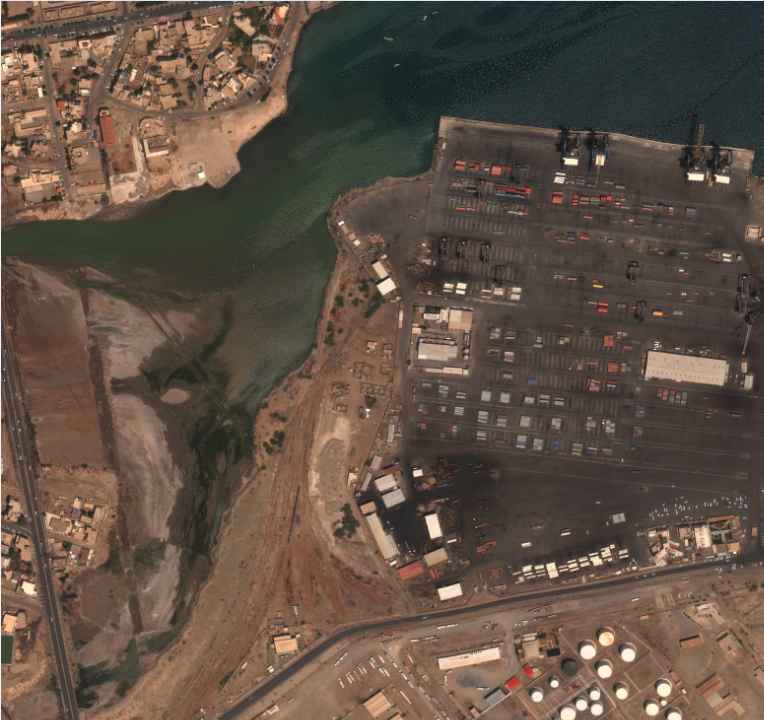
The discriminator takes in a 256x256 image from the generator and downsamples it by a half until its size is 8x8. Similar layers from the generator are used again in this discriminator such as the convolutional layers, the flatten, and dense layers. Despite these fully-connected layers, it’s still a DCGAN.

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 small pixel dimensions then gets deeper over time.

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

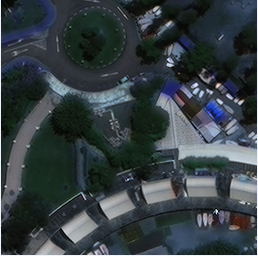
The training method implemented contains inputting the random noise vector, initializing the variable for fake and real images, and defining the hyperparameters. The entire dataset used is approximately 113,000 images which are cut up pieces or chipped pieces from the xView dataset. The batch size is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

1. Figures and Tables -



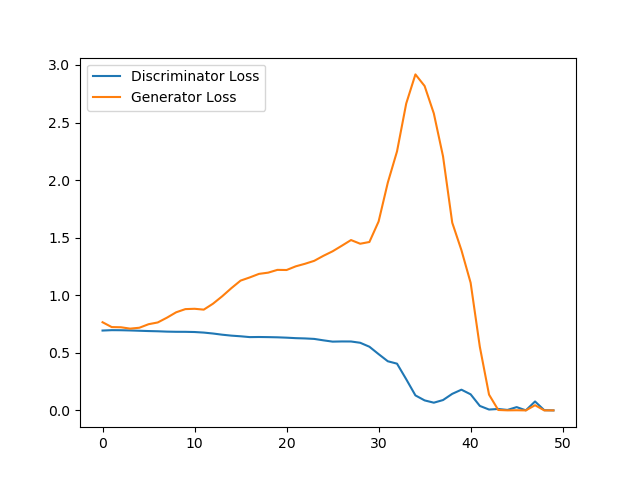
[Figure 1]

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 can be tagged and synthetically created with the synthetic images that the generator creates.



[Figure 2]

The image above (Figure 2), is a smaller image that was chipped from the larger images from the xView dataset.



[Figure 3]

The image above 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 that the models are following.

III. **Conclusion**

The Generative Networks can be trained directly and indirectly to either train a network to tell real data from fake one or vice-versa. The discriminator takes the data and tries to classify them as best as they can. GANs can be used in two ways: either by generating data that is realistic even though it is fake, or by creating the network to tell real data from fake ones.

The DCGAN starts off very slowly with a large image set for a while. On the other hand, the Progressive GAN starts with smaller layers of smaller pixel size of the image and slowly gets larger. These are much more accurate than a DCGAN.

In conclusion, GANs are very powerful neural networks that use the Generator and Discriminator to compete with each other to analyze the dataset that was placed. The GANs, despite the type used, generate synthetic images that the discriminator thinks are indistinguishable from the real image dataset.

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/>