USING GENERATIVE ADVERSARIAL NETWORKS TO AUGMENT UNMANNED AERIAL VEHICLE IMAGE CLASSIFICATION TRAINING SETS.

THESIS

Benjamin J. McCloskey, Second Lieutenant, USAF

AFIT-ENS-MS-22-M-151

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-22-M-151

USING GENERATIVE ADVERSARIAL NETWORKS TO AUGMENT UNMANNED AERIAL VEHICLE IMAGE CLASSIFICATION TRAINING SETS

THESIS

Presented to the Faculty

Department of Operations Research

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Benjamin J. McCloskey, B.S.

Second Lieutenant, USAF

March 24, 2022

AFIT-ENS-MS-22-M-151

USING GENERATIVE ADVERSARIAL NETWORKS TO AUGMENT UNMANNED AERIAL VEHICLE IMAGE CLASSIFICATION TRAINING SETS

THESIS

Benjamin J. McCloskey, B.S.
Second Lieutenant, USAF

Committee Membership:

Bruce Cox, Ph.D
Chair

Lance Champagne, Ph.D
Member

Trevor Bihl, Ph.D
Member

AFIT-ENS-MS-22-M-151

Abstract

A challenging task in computer vision is finding techniques to improve the object detection and classification capabilities of machine learning (ML) models used for processing images acquired by moving aerial platforms. Detection and classification of objects are usually accomplished through the application of supervised ML techniques, which require labeled training datasets. The collection of images for these training datasets is costly and inefficient. Due to the general impossibility to collect imagery from all possible elevation angles, sun angles, distances, and etc., this results in small training datasets with minimal image diversity. In an effort to increase the accuracy of supervised ML models trained on these datasets, various data augmentation techniques can be implemented to increase their size and diversity. Traditional data augmentation techniques, such as rotation and darkening of images, provide no new instances nor variety in the modified dataset. A Generative Adversarial Network (GAN) is a ML data augmentation technique which can learn the distribution of samples from a dataset and produce synthetic replications, known as "deepfakes." This research explores if GAN augmented Unmanned Aerial Vehicle (UAV) training sets can increase the generalizability of a detection model trained on said data. To answer this question, the You Only Look Once Version 4 - Tiny (YOLOv4-Tiny) Object Detection Model was trained with aerial image training sets depicting rural environments. The salient objects within the frames were recreated using various GAN architectures, placed back into the original frames, and the augmented frames appended to the original training sets. GAN augmentation on aerial image training sets led to a 6.75% increase on average in the Mean Average Precision (mAP) of the YOLOv4-Tiny Object Detection model with a best-case increase of 15.76%. Similarly,

a 4.13% increase on average and a best-case increase of 9.60% was observed for the Intersection over Union (IoU) rate. Finally, 100.00% True Positive (TP), 4.70% False Positive (FP), and zero False Negative (FN) detection rates were yielded, providing further evidence supporting GAN augmentation for object detection model training sets.

To my wife, thank you for all of your love, patience, and the endless amount of coffee you gave me to get me to the finish line of my thesis. I would not be here without you.

Acknowledgements

I would like to thank my advisor, Dr. Bruce Cox, for his guidance and mentorship not only throughout the course of my research but also for my future endeavors. Additionally, thank you to Dr. Lance Champagne and Dr. Trevor Bihl for their input and time in helping me complete my thesis. Finally, I would like to thank Dr. Michael Garee who gave up many hours of his time to help me learn, debug, and execute my computer coding.

Benjamin J. McCloskey

Table of Contents

Page

Page

Page

List of Figures

List of Tables

USING GENERATIVE ADVERSARIAL NETWORKS TO AUGMENT UNMANNED AERIAL VEHICLE IMAGE CLASSIFICATION TRAINING SETS

I. Introduction

The investigation of image and video classification techniques for data acquired from moving platforms is currently a growing area of interest in computer vision. Images collected by aerial vehicles are important for information gathering and gaining insights about an environment otherwise unattainable at a ground level evaluation. For training object detection models, one important characteristic of the training sets used for creating these models is the sets must contain a wide diversity of detail within their images. Past data augmentation techniques, for example rotating, adding noise, and flipping an image, were used to increase the diversity of a training set but were weak approaches due to their inability to add any new images to the dataset. Researching new image augmentation and classification methods which incorporate machine learning (ML) techniques can help lead to performance improvements in the models used for classifying aerial imagery.

1.1 Background and Problem Statements

1.1.1 Background

Recently, there has been an increase in usage of ML algorithms for classification or predictions on images. While ML has been used for many decades, on images it is the last 20 years for which we have seen reasonable progress. With the expansion of technological advancements in information gathering and storage as well as its

accessibility, the amount of data available for analysis is growing at an exponential pace. Increases in Random-access Memory (RAM) and hardware storage of computers have catered to the necessity for having enormous datasets for training, testing, and validating ML models to achieve lower bias and variance. Additional advancements in technology stem from improvements in a computer's Graphics Processing Unit (GPU) which allows for the ability to manipulate larger amounts of data at faster speeds, two important capabilities for real-time image processing [2].

Artifical Neural Networks (ANNs) are a subset of ML which are inspired by the biological structure of neurons in the brain directed towards solving complex classification and regression problems [3]. Deep learning is a subset of ANNs which create multiple interconnected layers in an effort to provide more computational advantages [3]. Convolutional Neural Networks (CNNs) are a subset of ANNs which allow for automated feature extraction and classification in unison. CNNs and ANNs in general require representative data to what is to be seen operationally, and thus often times they require enormous amounts of data due to the variations seen in the real world. While a plethora of data has been collected in the past decade, the problem of minuscule and imbalanced training data sets continues to hinder ML model training leading to poor, biased classification and analysis. Relatively small datasets lead to overfitting or underfitting in ML model training. Models which are overfit show promise in performance on the training data, but break down and cannot generalize to the associated real-world data fed to the model after it is trained. Overfitting a model can be averted by providing a larger and more diverse training dataset, as well as by reducing the complexity and introducing regularization into the model [4]. Models which underfit fail to learn the features and patterns of the training set and make inaccurate predictions on analogous real-world data. Increasing the complexity of the model can decrease the affects of underfitting. Another option to overcome

underfitting a model is to decrease the number of constraints placed upon the model [4].

There are many reasons why large, diverse image sets are useful for training models to detect objects captured in video frames. When video is taken from moving platforms, e.g. UAVs or cars, further issues exist as described by Bang et al. [5]. First, the time of day an image was taken as well as the conditions of the weather both which impact brightness and shadows. Second, the images gathered by moving platforms are sometimes blurred and distorted due to the type of camera being used and how it is affected by the physical vibrations projected from the moving platform's propulsion system. The moving platform's altitude, sun angle, look angle, cloud cover, and distance, as well as the color/shape/etc. of an object can further lead to distorted affects in the samples captured by the camera. The proclivity of researchers to ignore these parameters can lead to models which are susceptible to breaking down when faced with diverse operational data. These factors necessitate the need to gather large amounts of video frames containing various features, image irregularities, and distortions to replicate those found in real-world image collection so as to train a powerful object detection and classification model.

In order to increase image diversity in the hopes of improving the resulting accuracy of classification models trained on the data, the use of data augmentation can be implemented for distorting images collected by Unmanned Aerial Vehicle (UAV)s. A few current data augmentation techniques include flipping, rotating, or distorting the colors of an image. While these augmentation techniques may introduce more diversity in the dataset, they are unable to provide completely new frame instances for a model to be trained upon.

A Generative Adversarial Network (GAN) is a ML technique which learns from the probability distribution and features in a dataset to generate new synthetic in-

stances of the dataset, known as "deepfakes." The implementation of a GAN is a much more powerful data augmentation technique because it adds new, never before seen instances to a training set which are still plausible and representative of the originating population. Providing such new training instances to a ML model can lead to the model being more robust when used for detection in real-world operational environments.

1.1.2 Problem Statement

A common problem faced with image collection is not gathering sufficiently large and diverse training and testing datasets to produce efficiently performing ML models. The lack of diversity displayed by these minuscule training sets produce models performing poorly when used for real time detection. Finding ways to increase these datasets, whether it be through additional data collection or other methods, is important for creating a robust and generalizable model.

A second problem in computer vision is the insufficient increases in image diversity produced by traditional data augmentation techniques. Augmenting datasets through rotating, flipping, or darkening each of the gathered video frames fails to add any additional instances to the training set which compounds with the first problem noted above. The need to find a new data augmentation technique which provides new instances without the need to collect more data is important to quickly train detection models for deployment in rapidly shifting operational environments.

1.2 Research Questions

This research sought to answer the following questions:

1. Does an augmented image training dataset acquired by a moving platform containing generated synthetic images from a GAN increase the classification ac-

curacy and generalizability of a Convolutional Neural Network (CNN) object detection model?

2. Does an augmented image training dataset acquired by a moving platform containing generated synthetic images from a GAN increase the localization and generalizability of a CNN object detection model?

3. What inferences can be drawn from the unaugmented and augmented datasets that show their similarities and dissimilarities?

Providing evidence in support of the first and second questions could change how a data scientist conducts data collection and redirect their efforts towards using augmentation techniques with GANs to create datasets used for ML research. Not only should the model be able to classify an object but training a robust object detection model with the ability to find an object of interest in an image with a high Intersection over Union (IoU) value validates the model is capable of finding moving targets varying in their placement within a captured frame. The generalization of a model is the ability for the model to make accurate predictions and classifications on inputs never before seen by the network [6]. The augmented dataset must be similar to the original data set both qualitatively and quantitatively to justify assertions in model generalizability enhancement.

Answering the final question provides justification for whether the augmented objects from the GAN are similar in nature to the original samples and are reasonable replications for what would be found in a real-world environment. High rates of dissimilarity between objects of the same class could render GAN augmentation weak and in need of further research for practical usage.

1.3 Limitations of Research

One of the biggest limitations in this research was the accessibility to the proper hardware and software for implementing the different ML algorithms. While ML models can be executed on a Central Processing Unit (CPU), it would take the models in this thesis days, if not weeks, to run on a single CPU. A GPU is much more efficient when running deep learning models, especially those designed for image exploration. GPU access was very limited throughout the course of this research which added constraints to the amount of complexity in the CNN and GAN models as well as increasing the time needed to finish a training iteration for each of the models. Models could never be simultaneously ran, greatly increasing the timeline of completion for this thesis.

Another limitation was the amount of RAM and hard-drive memory available throughout the course of this research. Insufficient RAM further led to decreases in model complexity as well as the amount of data a model could utilize at a given moment in the training and testing processes of the research. Decreases in these two model components can lead to sub-optimal models. Steps were taken in this research to mitigate those affects including choosing models with less parameters but performed to the same high levels as more complex models. Additionally, partitioning the dataset into batches during training and testing helped alleviate RAM and hard-drive memory issues.

1.4 Thesis Organization

This chapter discussed the general area of ML this thesis will concentrate on, as well as outlined the benefits and limitations occurring in ML research. Chapter II provides a literature review which examines the theory of CNNs and GANs. In addition, it provides relevant research conducted using CNNs, GANs, and image frames

gathered from UAVs. Chapter III details the process of training the CNN detection model pre- and post- dataset augmentation. Chapter IV provides details of the generated synthetic objects used for augmenting the training sets. Chapter V presents the results gathered from the evaluation of the best models trained on the original and augmented training sets. Chapter VI outlines the methodology for three different experiments conducted after the conclusion of training with the original test set. The results of the three different experiments are reviewed in Chapter VII. Finally, Chapter VIII discuss the conclusions drawn from the results in addition to recommendations for future research in the data augmentation domain using Generative Adversarial Networks (GANs) for images acquired by moving platforms.

II. Background and Literature Review

This chapter provides information pertaining to the background of the different machine learning (ML) techniques performed in the research. Additionally, relevant research which provided motivation for the thesis methodology is also examined. The main topics explored are Artifical Neural Networks (ANNs), Convolutional Neural Networks (CNNs), Generative Adversarial Networks (GANs), and the object detection model You Only Look Once Version 4 - Tiny (YOLOv4-Tiny).

2.1 Artificial Neural Networks

2.1.1 History

ANNs are a ML model architecture inspired by the structure of the brain and its path of neurons which allow for human intellectual thinking. Dating back to 1943, McCulloch and Pitts developed an early rendition of a mathematical model which represented a biological neuron [7]. The neuron accepted one or more binary inputs, leading the formulation of a binary output dependent upon how many inputs were activated.

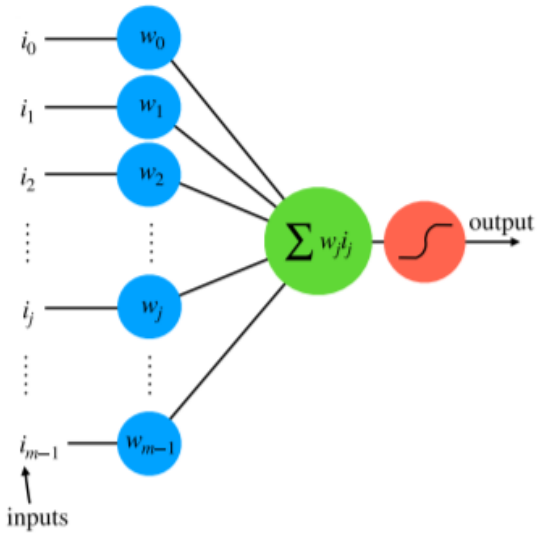


Figure 1: Artificial Neuron (Image from [8])

The first Artificial Neural Network (ANN), shown in Figure 1, was a single layer perceptron created by Francis Rosenblatt in 1957 [9]. Rosenblatt's structure shifted from only accepting binary inputs and outputs to numerical values with the use of a Linear Threshold Unit (LTU). A LTU takes the sum of the input values multiplied by their respective weights

$$z = w_1 x_1 + w_2 x_2 + ..w_n x_n = \mathbf{W}^T \mathbf{X} \tag{1}$$

where the calculated value is then passed through a step function for classifying the collection of inputs as "0" or "1". The Heavyside step function

$$Heavyside(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 1 \end{cases} \tag{2}$$

was popular in early neural network training.

One problem with the singe layer perceptron was how it was incapable of solving

nonlinear problems. This issue was resolved with a Multilayer Perceptron (MLP), also known as a feed forward ANN. A MLP is a stack of multiple perceptrons, or one or more layers of LTUs. Improvements in the MLP were made when Rumelhart et al. [10] refined the backpropogation method of updating the model's weights by shifting the step activation function to a nonlinear activation function. The step function was replaced with a Logistic Activation Function, also known as the Sigmoid Activation Function

$$\sigma(z) = \frac{1}{1 + \exp -z} \tag{3}$$

where this activation functions differs from a step function by adopting a shape containing no flat sections. The curved function has a nonzero derivative which can always be found using gradient descent during backpropagation to update the network's weights.

2.1.2 Architecture

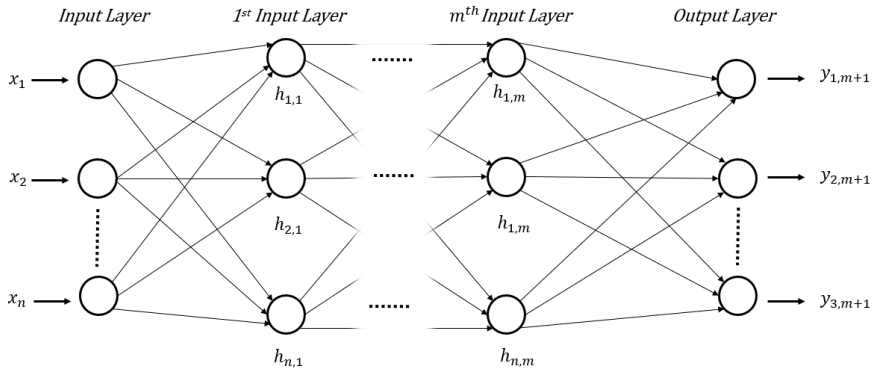


Figure 2: Artificial Neural Network

2.1.2.1 Input Layer

Portrayed in Figure 2, the input layer directly accepts the samples which are needed to be processed by the ANN. The number of neurons in this layer depend on the number of features in the samples of the dataset. The size of the input layer does not need to be the same size of the output layer.

2.1.2.2 Hidden Layers

In between the input and output layer are a series of hidden layers containing Linear Threshold Units (LTUs). A neural network is considered a Deep Neural Network (DNN) if it contains more than one layer of LTUs [4]. Bias neurons, which have a value of "1," can also be added to each hidden layer for input into the next layer.

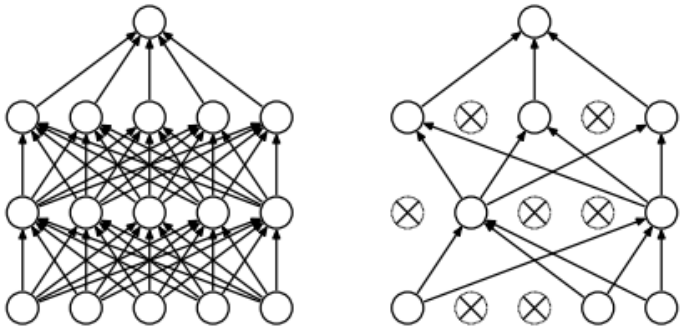
2.1.2.3 Output Layer

The output layer provides the prediction or classification by the neural network. For a binary classification network, the output will be congruous with one of the two binary classes. Utilization of a Softmax Activation Function allows for the output to provide classification probabilities corresponding to more than two classes.

2.1.2.4 Dropout

One problem occurring in neural network training is overfitting of the model. As stated in Chapter I, an overfit model will show promising performance on the training dataset but breaks down and has low generalizability on the associated real-world data fed to the model post training. One way to overcome the presence of overfitting in a model is dropout. Dropout, presented in Figure 3, will randomly drop neurons in the layers of a neural network during the course of the model's training. Placing dropout in the architecture of a neural network helps overcome two problems

in model training: the need to create many differently trained models for different ML analytical tasks and the requirement of having large amounts of data to efficiently train a neural network [11]. By dropping neurons during model training, various "thinned" architectures can be trained for one model. Creating a model capable of performing well with less neurons leads to better performance on the testing and validation data when it is given the ability to once again use all of the neurons within its hidden layers, promoting a more generalized model. It also creates faster and more reliable model performance [12].



(a) Standard Neural Net (b) After applying dropout.

Figure 3: Example of Dropout (Image from [11])

2.1.2.5 Backpropagation

The success of ANNs, in addition to other neural network architectures, can be attributed to the development of backpropagation in neural network training. First introduced by Werbos [13] in 1974, backpropagation became well know when Rumelhart et al. [10] added a nonlinear activation function to the process and showed the effectiveness of the gradient descent technique. Backpropagation used in tandem with an optimizer is implemented in training neural networks for updating their weights. Werbos later described the goal of backpropogation as updating the weights of the model in a formulation which produces output predictions, $\hat{y}$, close to or the same

as the original data samples, y [14]. The initial weights of a neural network can be randomly selected at the first epoch of training. After completion of the first epoch, the calculated $\hat{y}$ is inspected and the error for the model's weights is calculated. Next, using the chain rule, the derivative with respect to the weights for the error is calculated. Weights are either increased or decreased depending on which direction shift will decrease the output error. A learning rate can be attached to the update in weights to speed up or slow down the process. The final weight after back propagation is the difference between the original weight and the product of the learning rate for each specific weight's ordered derivative. Using the gradients from the loss function, a model can properly update weights via a gradient based step.

2.1.2.6 Optimizers

Once the gradients of a network are calculated through backpropogation, an optimizer can be used to update the model's weights to efficiently align with the decision boundaries of the problem task. The Root Mean Squared Propagation (RMSProp) Optimizer

$$\begin{aligned} &1.\ s \leftarrow \beta s + (1-\beta) \bigtriangledown_\theta J(\theta) \otimes \bigtriangledown_\theta J(\theta) \\ &2.\ \theta \leftarrow \theta - \eta \bigtriangledown_\theta J(\theta) \oslash \sqrt{s+\epsilon} \end{aligned} \tag{4}$$

uses exponential decay and only recently acquired gradients for updating the model [4]. The first step exploits exponential decay and adds the squares of the gradients to vector s. β is a momentum term value between zero and one which adds "friction" to the gradient used to accelerate the updating of weights in model training. The second step updates the weights θ by taking the difference of the gradient of the cost function with respect to the weights, $\bigtriangledown_\theta J(\theta)$, multiplied by the learning rate η and

the weights. $\sqrt{s+\epsilon}$ scales down the gradient by a factor where ϵ is a smoothing term to ensure there is no division by zero.

The Adaptive Moment Estimation (Adam) Optimizer

$$
\begin{aligned}
&1.\ m \leftarrow \beta_1 m + (1-\beta_1) \bigtriangledown_\theta J(\theta) \\
&2.\ s \leftarrow \beta_2 s + (1-\beta_2) \bigtriangledown_\theta J(\theta) \otimes \bigtriangledown_\theta J(\theta) \\
&3.\ m \leftarrow \frac{m}{1-{\beta_1}^T} \\
&4.\ s \leftarrow \frac{s}{1-{\beta_2}^T} \\
&5.\ \theta \leftarrow \theta - \eta m \oslash \sqrt{s+\epsilon}
\end{aligned}
\tag{5}
$$

is predominately used today and combines momentum optimization with features of the RMSProp Optimizer [4]. The first step of the equation shows how the Adam Optimizer adopts momentum optimization which tracks the exponentially decaying average of past gradients [4]. In step one, the local gradient is added to m, the momentum vector. Step two is equivalent to the exponential decay used in RMSProp. Steps three and four help "boost" m and s when training begins since both of those values begin at zero [4]. Step 5 is similar to Gradient Descent and updates the weights and T is the current iteration.

2.1.2.7 Vanishing/Exploding Gradients

The vanishing/exploding gradient problem can occur during the backpropagation phase of the network's training. These terms refer to when a model has become saturated with tiny or enormous weights. "Vanishing" refers to gradients which continuously become substantially smaller as the backpropogation algorithm moves to lower layer calculations [4]. The infinitesimal or zero-value gradients create little to no change in the updated weights. "Exploding" refers to when the gradients become

so large or even reaching a "NaN" computer error the huge changes in the weights lead the algorithm to diverge.

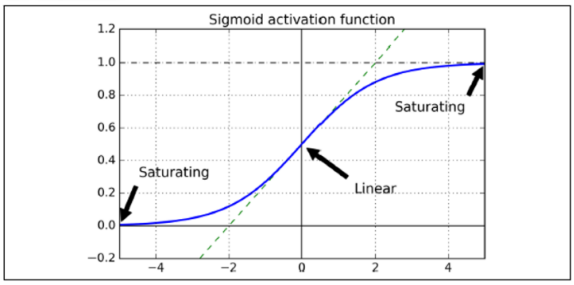


Figure 4: Example of Saturation in the Sigmoid Activation Function (Image from [4])

While activation functions help with overcoming the vanishing/exploding gradient problem, saturation still frequently can occur with the use of the Sigmoid Activation Function.

2.2 Activation Functions

One problem early neural network architectures had was their structure being linear in nature and failing to solve nonlinear problems. This apparent lack of versatility led to stagnation in the development of ANNs for many years. While the Sigmoid Activation Function is useful in some circumstances, its tendency to saturate with incorrect initialization causes instability in the training of different variations of neural networks [15]. Activation functions add non-linearity in the feedforward process of the network as well as backpropogation adjustments leading to more effectively updated weights which form to the decision boundary of the data. Selecting the right activation function helps overcome the "vanishing/exploding gradient" problem and can give rise to a decrease in the time needed for training a robust neural network.

The Rectified Linear Unit (ReLU) Activation Function has become the current default activation function within the inner layers of a neural network due to its ability to overcome saturation. Improvements in training stem from the ReLU Activation

Function sending any negative input in a hidden neuron to zero and returning any positive output falling between zero and the maximum, encouraging sparsity in the hidden neurons. In addition, convergence speed increased since the ReLU has no exponential or division properties in its operation [16]. In the calculation

$$f(x) = max(0, x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{6}$$

any value which is zero or negative for the ReLU Activation Function will go to zero. By setting the value of negative numbers to zero, the vanishing/exploding gradient problem can be averted in backpropogation.

Gloret et al. [17] showed the capability of the ReLU Activation Function in introducing and increasing sparsity from 50% to 80% in a neural network. Sparsity in a model is important because it gives neurons real values or zero for firing to the next layer. This benefits a neural network by untangling messy features which explain the data, inducing the model to generalize more efficiently to small changes in input instances. Other factors introduced by sparsity to a model are allowing for the variation of active neurons in each layer. Active neuron variation strengthens a model's understanding towards samples of different shapes exhibiting varying amounts of information and leads to less non-linearity in the information pushed through the hidden layers since sparse representations of data are generally more linearly separable [17].

While the ReLU Activation Function elicits major improvements in nonlinear transformations, problems still arise from the function's weaknesses pertaining to nonexistent gradients and information loss for values below zero. Maas et al.[18] reconstructed the ReLU Activation Function by adding a slope value to its negative side instead of sending negative values to zero. The Leaky Rectified Linear Unit

(LReLU) Activation Function

$$f(x) = max(0, x) = \begin{cases} x & x > 0 \\ 0 & mx \leq 0, \text{where } m \text{ is a preselected slope value} \end{cases} \tag{7}$$

has an additional slope parameter m. In the paper by Maas et al, the authors used a value of 0.01. The small, nonzero gradient helps overcome the saturation occurring in the ReLU. To investigate the performance of the LReLU, the researchers performed an experiment on acoustics data used for speech recognition systems. In their experiment, three different activation functions, Hyperbolic Tangent (tanh), ReLU, and LReLU, were used in a DNN and their performances compared. While the performance of the LReLU Activation Function closely mimicked the ReLU Activation Function's performance, both showed to more effectively increase model sparsity compared to the tanh Activation Function.

Another metric the authors used for assessing the performance of each activation function was dispersion. Dispersion [19] is the measurement of the differences in each set of activations when shown a different stimulus [18]. The standard example explaining dispersion is four units coded with probabilities of $[1, 0, 0, 0]$. These will have the same average sparsity of 25% as four coded units with probabilities $[0.25, 0.25, 0.25, 0.25]$. The first set is an example of a compact coding while the second set is an example of sparse-dispersed coding. Sparse-dispersed coding is more beneficial for noisy inputs which are more prominent in imagery data and does not hinder neural networks constrained by the pinpoint accuracy, a necessity for compact coding sparsity [19]. By measuring the standard deviation of activation probabilities for all of the units in a hidden layer, Maas et al. could measure how effective an activation function was in creating sparse outputs. If the standard deviation is zero, all of the units will be coded equally. Both rectified activation functions yielded stan-

dard deviations of 0.04 while the tanh Activation Function had a standard deviation of 0.14. The researchers concluded that rectified activation functions produce sparse codes more uniformly distributed across a hidden layer compared to the previous standard Sigmoid-style activation functions.

The Swish Activation Function is a non-monotonic activation function capable of maintaining small negative weights [20], unlike the ReLU. Additionally, the Swish Activation

$$f(x) = xsigmoid(\beta x) \tag{8}$$

has a smoother shape which helps with backpropagation where β is a constant parameter chosen by the user. The first derivative of the Swish is the Mish Activation Function, introduced by Diganta Misra [20]. The Mish Activation Function

$$f(x) = x \tanh\left(\ln(1 + e^x)\right) \tag{9}$$

is a self regularized non-monotonic activation function and is effective when embedded in computer vision models.

Mirsa compared the effectiveness of the Mish Activation Function to the ReLU and Swish Activation Functions. The first experiment conducted used various neural network architectures for classifying the CIFAR-10 [21] dataset. For the CIFAR-10 dataset, every network utilizing the Mish Activation Function produced levels of accuracy 1.0% to 3.0% higher compared to the same architectures adopting the Swish and ReLU Activation Functions. Another experiment used the Microsoft Common Objects in Context (MS-COCO) [22] dataset to investigate how the Mish Activation Function performed when placed in object detection models. The metric analyzed was mean average precision at IOU threshold .5 (mAP@.50). The first object detection

network used was CSP-DarkNet-53 [23]. The Mish Activation Function increased accuracy by 0.4% compared to the model supported by the ReLU Activation Function. An additional object detection experiment compared the LReLU and Mish Activation Functions placed within the You Only Look Once Version 4 (YOLOv4) Object Detection Model. Investigating three different YOLOv4 architectures, Mirsa provided results which showed the Mish Activation Function producing mAP@.50 values which were 0.9% to 2.1% higher than the networks adopting the LReLU. These findings suggest the Mish Activation Function may lead to better performance when used in an object detection network's architecture. The Mish Activation Function consistently led to enhanced performance for models trained and designed for image classification and detection.

2.3 Convolutional Neural Networks

CNNs are an extension to the ANN framework which have shown great success for image recognition and classification. Albawi et al. [24] characterized a few features which make the Convolutional Neural Network (CNN) architecture unique and advantageous for image inspection compared to other neural network architectures. The CNN architecture is designed to have significantly less parameters than a traditional ANN. Each convolutional layer in a CNN performs the matrix convolutional operation on the numerical representation of an image's pixels which allows for the network to recognize patterns and find important features in given sample. Other important features in a CNN include its pooling layers for dimensionality reduction, its flatten layer for transforming a two-dimensional image array into a one-dimension, and its fully connected layers which mirror an ANN and result in the classification or prediction of a given sample.

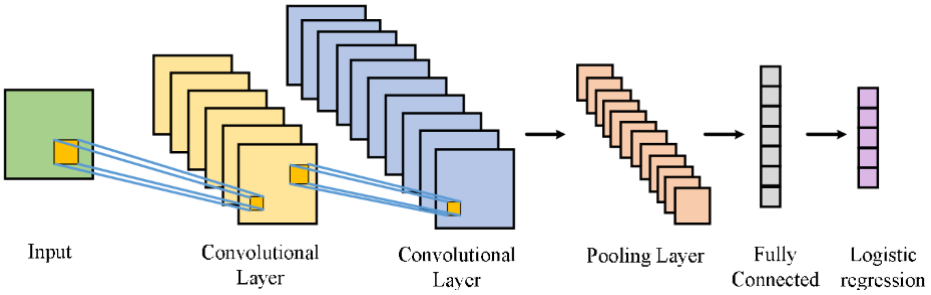


Figure 5: Convolutional Neural Network (Image from [25])

2.3.1 History of the CNN

In 1980, Fukushima et al. proposed an early rendition of a CNN, called the "Neocognitron", and showed its capability to achieve high classification accuracy on a handwritten image dataset [26]. LeCun et al. built upon Fukishima's work in 1989 with a two-dimensional CNN [27]. The research from LeCun et al. showed how using backpropogation [28] automates the training for a CNN and leads to its ability to more effectively classify image datasets. In 1998, LeCun et al. [29] made improvements to the 1989 architecture by making the network deeper which reaped more efficient classification capabilities. Their network, called Le-Net5, showed major improvements in neural network training with its decrease in memory usage and increase in training speed. Le-Net5 showed the ability of CNN classification models to analyze images of higher qualities and extract important latent features from the sample.

After Le-Net5, improvements in CNNs went stagnant for over a decade until 2012 with the release of AlexNet [30]. AlexNet has an architecture containing five convolutional layers followed by three fully connected layers with sixty million tunable parameters and 650,000 neurons. The hidden layers use the ReLU Activation Function while the final activation function is a Softmax Function with 1,000 possible prediction outputs. Due to its size, AlexNet had to be trained on two Graphics Processing Unit (GPU) for five to six days. The deep architecture of AlexNet was able to

yield much lower error results on the two ImageNet datasets [31], with 37.5% for the Top-1 ImageNet dataset and 17.0% for the Top-5 ImageNet dataset, much better than the second best error rates of 45.7% and 25.7%. AlexNet showed how advancements in computational technology as well as diversity, complexity, and size of datasets are factors which have allowed CNN classification models to evolve into high power ML frameworks in the 21^{st} century.

In 2014, CNNs expanded in the computer vision domain for object detection with the improvements in architecture and creation of the Regions with Convolutional Neural Networks (R-CNN) by Girshick et al. [32]. Object detection incorporates two domains of image inspection, combining classification of the object with localization for segmenting an image apart and highlighting the salient features of interest. The act of locating where a salient object is within an image is called *localization*.

The first step of the R-CNN is to create category dependent region proposals. To generate various region proposals for locating different objects in an image, selective search is used which segments various objects in a frame apart from each other. Uijlings et al. [33] proposed a method which many R-CNNs have incorporated into their architectures. Using *bottom-up grouping*, small segments can be partitioned across an image and slowly combined based of the color, texture, size, and fill similarities between the segmented sections. Once the segmentation of regions is complete, a fixed length feature vector is created and fed into the CNN. Girshick et al. used a CNN architecture designed with five convolutional layers and two fully connected layers. Finally, Support Vector Machines (SVM) trained on particular classes of the dataset scored the features extracted, which showed the neural networks were able to find the features of interest. Their work outlined improvements in the segmentation of images and provided support for the proof of concept asserting classification models need more labeled data rather than more localization information. This enables

transfer learning to be used with pretrained classifiers to acquire high level results for imbalanced and small datasets. These improvements have pushed CNNs to evolve from models used strictly in either deep learning or computer vision to models now spanning the two fields, combining classification and localization in conjunction with each other for solving complex image analysis problems.

2.3.2 Architecture

2.3.2.1 Convolutional Layers

The power of a CNN derives from the convolutional layers which are able to target and extract information from the densely correlated points of an image's features [34]. The number of neurons in a convolutional layer rely on the computing power, memory, and complexity of the dataset containing the information being extracted. A convolutional layer performs the matrix convolution operation [34]

$$f_l^k(p,q) = \sum_c \sum_{x,y} i_c(x,y) e_l^k(u,v) \tag{10}$$

on the given tensor. The convolutional operation takes a segment $i_c(x,y)$, of the input image I_c and conducts element wise multiplication between the image segment and $e_l^k(u,v)$, where k_I is the k^{th} convolutional kernel indexed by the I^{th} layer. The area of the image where data is being extracted from is given by the coordinates x and y. c is the index of the channel, and there are typically three channel indices for Red, Green, Blue (RGB) images and one channel index for grayscale images. The output at the k^{th} convolution is a feature map

$$\mathbf{F}_l^k = [f_l^k(1,1),,,f_l^k(p,q)...f_l^k(P,Q)] \tag{11}$$

where P is the total number of rows of a feature matrix and Q is the total number of columns of a feature matrix.

2.3.2.2 Convolutional Filters (Kernels)

The filters in a convolutional layer, also distinguished in some texts as kernels, break the input into the layer down by placing different weights on each neuron to produce various feature maps. Neurons will ignore all information except for the information gathered in their receptive field. For example, a neuron with a kernel representing a horizontal line through the middle of its nxn receptive field will multiply all inputs by zero except for the inputs crossing the centered horizontal line. The movement of the convolutional kernels across an image are dictated by the layer's stride. The stride in a convolutional layer is the distance between two receptive fields and is a hyperparameter which sets how many pixels a kernel must move between each receptive field.

2.3.2.3 Padding

Since the input size of a sample is reduced as it moves through the layers of a CNN, padding ensures that the output shape of the network is the same as the input shape. Zero padding will add zeros to any boundary of the input image which is dropped when passing down into the layers of a CNN.

2.3.2.4 Pooling Layer

Following a convolutional layer, the pooling layer's purpose is to perform dimensionality reduction on the image, reducing the number of parameters and the model's complexity [35]. In the pooling layer, "outputs of several nearby feature detectors are combined into a local or global 'bag of features', in a way that preserves task-related

information while removing irrelevant details," [36]. The goal of the pooling layer is to acquire a sub-sample of the image at each step and uncover the invariant features [37]. Two methods which are popular in image classification are average pooling and maximum pooling. Average pooling takes the average of all elements inside a kernel at each stride step.

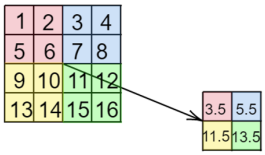


Figure 6: Average Pooling Operation

Maximum pooling takes the maximum number inside a filter at each stride step. For object recognition, maximum pooling has been found to decrease error and provide improvements in image downsampling for feature extraction.

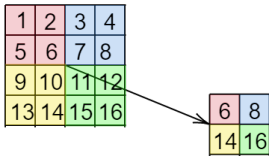


Figure 7: Max Pooling Operation

Scherer et al. [38] experimented with subsampling pooling and maximum pooling in CNN architecture to see how the different pooling techniques affected an object recognition model's performance. Average pooling is a subset of subsampling pooling. One difference with their subsampling pooling layer and an average pooling layer occured after taking the average of the inputs, the average was multiplied by a trainable scalar and then was fed through a tanh Activation Function. Experimenting on two different image recognition datasets, their results showed the max pooling operation producing lower error rates for the classification model than the subsampling pooling operation. For example, on the Caltech-101 image dataset [39], using the subsampling

pooling operation in the pooling layers produced a model with an error rate of 65.9% while using the max pooling operation in the pooling layers lowered the error rate to 55.6%. Additionally, for the NORB image recognition dataset [40], the utilization of the subsampling pooling technique produced a model with an error rate of 7.32% (±1.27%) while the max pooling technique had again yielded a lower error with a rate of 5.22% (±0.52%).

2.3.2.5 Fully Connected Layers

Once an image is passed through the convolutional layers of a CNN architecture, it must be flattened into a one-dimensional array to go through a series of fully connected layers. The fully connected layers of the CNN architecture mirror a typical ANN. The information is passed through the layers and finally into an activation function. Back propagation is then used with support from the network's optimizer for updating the fully connected layers' weights to output more accurate results.

2.4 You Only Look Once Version 4 (YOLOv4)

YOLOv4 is a CNN object detection model first introduced by Bockvoskiy et al.[23] in 2020. It is a variant of the You Only Look Once Version 3 (YOLOv3) Object Detection Model with additional layers and feature aggregation techniques, increasing the complexity but also the accuracy of the model compared to YOLOv3.

2.4.1 Architecture

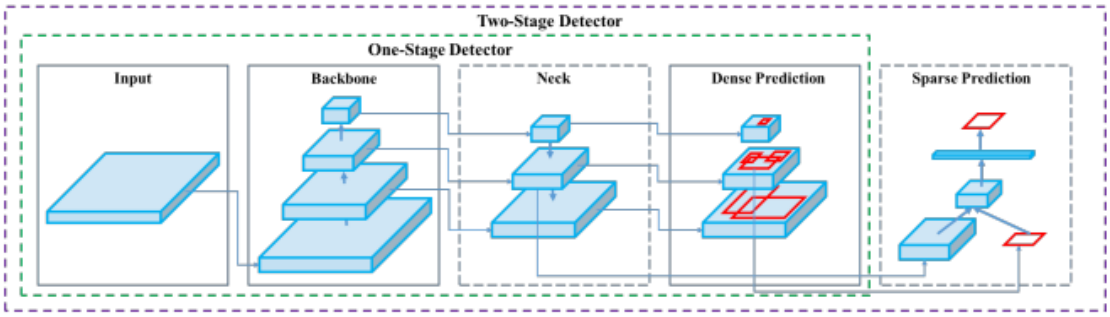


Figure 8: Object Detector Framework

As outlined by the authors, object detection models are comprised of various segments. Shown in Figure 8. these sections include the backbone, neck, and head. The backbone of the YOLOv4 architecture is CSPDarkNet-53 [23]. The neck of the architecture utilizes Spatial Pyramid Pooling (SPP) [41] and a Path Aggregation Network (PANet) [42]. Finally, the architecture ends with the head which adopted the same head as the predecessor model YOLOv3 [43].

2.4.1.1 Backbone

The backbone in an object detection model extracts the features from an image for the detection model to analyze [44]. The backbone of YOLOv4 is DarkNet-53, the image classifier, and a Cross Stage Partial Network (CSPNet), a method which splits and passes unreduced feature maps to the next transition layer. DarkNet-53, outlined by Figure 9, was adapted from DarkNet-19 and first implemented by Redmon et al. [43] in their creation of the YOLOv3 Object Detection Model.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 9: Summary of DarkNet-53 from Redmon et al. [43]

The 53 convolutional layers end with the average pooling operation, a 1000 neuron fully connected layer, and a Softmax Activation Function to output the predicted class for each object of interest. YOLOv4 is changed from this architecture with the transplanting of the YOLOv3 head over the final layers.

CSPNet [45] was developed by Wang et al. to reduce the amount of computations required for a CNN by segmenting feature maps at the base layer and then further unifying them back together through a cross-stage hierarchy. Their network was formulated on the idea that the gradient flow could be split and disseminated through

different network paths. The authors evaluated their architectural backbone and its ability to solve three problems which are prominent in CNNs. These problems were strengthening the learning ability, removing computational bottlenecks, and reducing the memory costs of a CNN [45]. As stated by the authors, "CSPNet separates feature map[s] of the base layer into two part[s], one part will go through a dense block and a transition layer; the other one part is then combined with [the] transmitted feature map to the next stage," [45].

For supporting the learning ability of CNNs, the authors applied the CSPNet to ResNet [46], ResNeXt [47], and DenseNet [48]. The amount of computational power decreased from 10% to a 20% decrease in conjunction with classification accuracy increasing for the ImageNet dataset [31]. For the problem of computational bottlenecks, the use of a CSPNet reduced bottleneck effects by 80% for models based off of the YOLOv3 architecture. Finally, for mitigating the problems of limited memory usage, adapting a CSPNet showed significant decreases in the amount of memory needed. Wang et al. provided their results when executing PeleeNet [49] using CSPNet showing computational memory usage for the model to produce its feature pyramids decreased by 75%.

The backbone incorporates different techniques which have no effect on the inference cost but increase the accuracy of the model, defined by the authors as "Bag of Freebies (BoF)" [23]. These techniques include CutMix and mosaic data augmentation, DropBlock regularization, and class label smoothing. CutMix [50] crops a portion of one image and adds the segmented portion to other images. Mosaic augmentation builds on the CutMix technique by combining four images together. DropBlock [51] is analogous to the dropout technique used in CNNs except instead of dropping single neurons at each layer, entire feature maps are dropped. Label smoothing removes one hot coding for classes, "using soft targets that are a weighted

average of the hard targets and the uniform distribution over labels," [52].

"Bag of Specials (BoS)" is defined by the authors as the methods used post processing which create small increases in the inference cost but lead to huge improvements in the performance accuracy of the object detection model. In the backbone, these include Multi-input weight residual connections (MiWRC) in addition to the previously defined Mish Activation and CSPNet. MiWRC add feature maps of different scales through a scale-wise level re-weighting.

2.4.1.2 Neck

Once the backbone extracts the features from a sample, the neck collects feature maps from different phases of the backbone and aggregates them in preparation for predictions by the head. The neck of the YOLOv4 Object Detection Model is guided by SPP [41] and PANet [42]. The SPP operation abolishes the fixed input originally required by a CNN and improves the receptive field. For object detection, a SPP-net pulls out "window-wise" features by first performing a pooling operation of all of the feature maps in a given image into spatial bins. Since the sizes of the spatial bins are proportional to the size of the input image, they will stay fixed even with varying image sizes. SPP is then used to take sub images of the feature maps which result in fixed length instances to be passed to the fully connect layers, presented in Figure 10. As mentioned by the authors, this decreased the time for a CNN to train and run since the convolutional operations are only exercised once on an image. Bockvoskiy et al.[23] gained significant increases in the receptive fields and extraction of important features without a reduction in YOLOv4's speed when adapting SPP over CSPDarkNet-53.

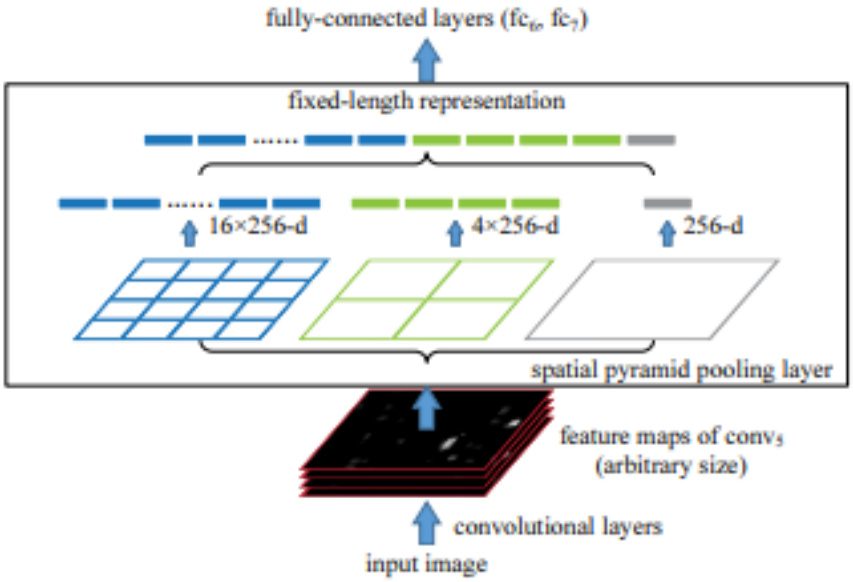


Figure 10: Example of Spatial Pyramid Pooling Operation from He et al. [41]

A PANet shortens the path of lower level layers to the highest level of a feature path in a segmented instance by adopting bottom-up path augmentation. As stated by Liu et al. [42], using a PANet backbone will also smooth the paths where feature information is passed through an aggregate pooling operation, pooling together the features from all feature levels.

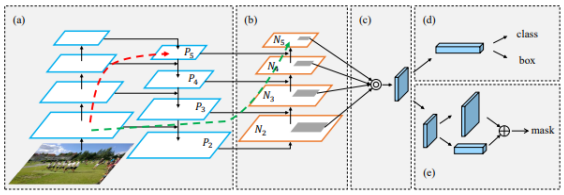


Figure 11: Path Aggregation Network Framework from [42]: (a) Feature Pyramid Network Backbone. (b) Bottom-up Path Augmentation. (c)Adaptive Feature Pooling. (d) Box Branch (e) Fully-connected Fushion.

Figure 11 outlines the PANet framework. Feature Pyramid Network (FPN) will extract features from the images, but some of the features at the input of the image will be passed to the Bottom-Up Path Augmentation step. This step shortens the

path required for information to be passed and enhances the feature pyramid using the low-level localization information from the features. Adaptive Feature Pooling is a feature aggregation step which helps revive the fragmented information path and combines features at all levels for each proposal. The last step is a combination of an augmented mask prediction with fully connected layers which make box predictions and classifications.

2.4.1.3 Head

After the backbone extracts the features of interest and the neck aggregates those features together, the head of the object detection network executes bounding box regression and classification predictions on the samples. YOLOv4 transfers over the YOLOv3 [43] structure for the configuration of its head. Three different outputs emanate from the YOLOv3 head. First, a three-dimensional tensor coded bounding box which is predicted by anchor boxes based off four bounding box coordinates. After the bounding box is predicted, an objectness measurement is then predicted. Objectness makes a prediction of the Intersection over Union (IoU) which is the difference between the ground truth box and the predicted box (see equation in Chapter III). Finally, a prediction of the conditional probability of a class is made on the object of interest encompassed within the predicted bounding box.

2.4.1.4 Additional Bag of Freebies/Bag of Specials

The detector adopts many BoF for regularization. These techniques include Complete Intersection over Union (CIoU) loss, Cross mini-Batch Normalization (CmBN), DropBlock regularization, mosaic data augmentation, Self-Adversarial Training (SAT), eliminate grid sensitivity, the use of multiple anchors for a single ground truth, cosine annealing scheduler, optimal hyperparameters, and random training shapes. CIoU

loss is a ground loss for bounding box regression which accounts for overlapping area, central point distance, and aspect ratio of ground truth and predicted bounding boxes [44]. CmBN builds upon the batch normalization technique by breaking each batch into four additional sub-batches. Statistical information is only gathered between the four sub-batches of each regular batch. SAT is conducted with two forward backward stages. During stage one, none of the neural network's weights are affected, but rather the original images are modified. The goal of this stage is to distort the image in a manner which makes it difficult for the network to find the objects of interest. In stage two, the neural network will conduct object detection on the distorted image. Eliminate grid sensitivity is used to assess the object coordinates. Using multiple anchors for a ground truth helps with achieving detection at a certain IoU threshold. The cosine annealing scheduler alters the learning rates for any of the processes using the Sigmoid Activation Function [53]. The hyperparamters for the model were chosen by genetic algorithms. Random training shapes increase the size of mini batches during the smaller resolution stages in model training. Finally, DropBlock regularization and mosaic data augmentation were implemented in the same manner as previously defined.

The BoS within the detector include the Mish Activation Function, SPP-Block, Spatial Attention Module (SAM)-block, PAN path-aggregation block, and Distance Intersection over Union (DIoU) non-maximum supression (NMS). The Mish Activation Function, as previously stated, bolsters the performance of models conducting computer vision tasks. As originally asserted, SPP increases the receptive field and segments out the most important features. SAM creates a spatial attention map of the most revealing features in a sample which are inter-spatially related [54]. Characteristics of a PANet are used for aggregating features from different levels of the backbone. Finally, DIoU NMS accounts for the center points of two bounding boxes

when repressing superfluous bounding boxes.

2.4.2 Performance

One important feature of YOLOv4 is its ability to be trained on a single GPU. This makes its use desirable for a setting with limited resources, a common constraint for ML problems. The results in Figure 12 from Bockvoskiy et al.'s research shows the evaluation for the YOLOv4's performance compared to other state of the art object detection models as of 2020. The findings conclude that YOLOv4 could be trained quicker with higher levels of accuracy versus the other models in the test.

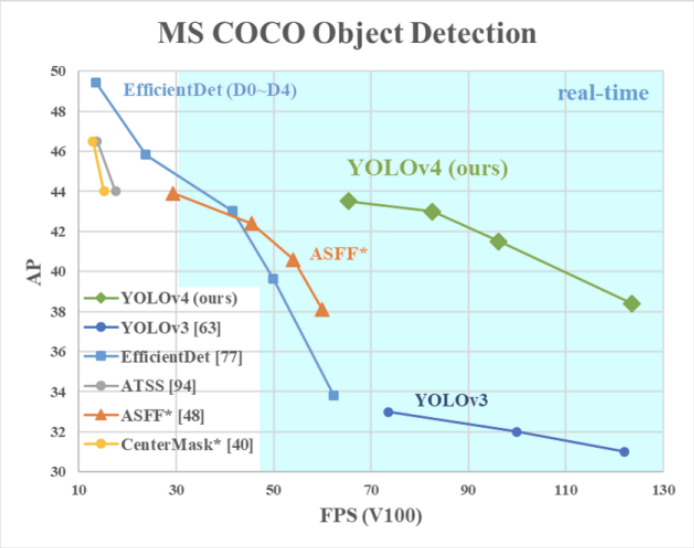


Figure 12: The performance of YOLOv4 compared to other models. [23]

Compared to its predecessor YOLOv3, YOLOv4 had a 10% improvement in Average Precision (AP) and a 12% improvement in Frames Per Second (FPS).

2.5 YOLOv4-Tiny Configuration

One complication in deploying detection models on moving platforms is limited available on-board memory. The scaled YOLOv4 Model mitigates this constraint with

smaller memory requirements. Designed by Wang et al. [55], the tiny architecture was able to achieve FPS rates of an upwards 1774 frames, making the configuration a strong candidate for real time detection on smaller devices. The authors also compared the proposed YOLOv4-Tiny architecture to other known small detection models and showed their architecture achieving the highest AP on the MS-COCO dataset after 600 epochs. Their model achieved a score of 28.7% which was approximately 5% higher than the next best model, the ThunderS146 [56], which achieved an AP of 23.6%. YOLOv4-Tiny is reduced in size from the original 137 trainable layers to 29 trainable layers. Instead of ending with three YOLO layers at the end of its network architecture, the YOLOv4-Tiny configuration only contains two.

2.6 Generative Adversarial Networks

2.6.1 History

In 2014, Goodfellow et al. [57] proposed the first Generative Adversarial Network (GAN) which consisted of a generator for producing synthetic data and a discriminator for detecting if a sample was artificial or real. The generator and discriminator would "compete" back and forth towards Nash Equilibrium using backpropagation with an optimizer to update weights during the training process in pursuit of producing high quality synthetic data.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}(x)[\log D(x)] + \mathbb{E}_{z\sim p_z}(z)[log(1 - D(G(z)))] \tag{12}$$

DThe original GAN architecture possessed the value function outlined by Equation (12). The goal of the discriminator, as described by the original paper, is to detect whether a sample is "fake" or "real." Equation (13) shows the goal of the discriminator which is to maximize the log-likelihood conditional probability $P(Y = y|x)$. If

y comes from the real dataset, y equals one. If y comes from the generator, y equals zero.

$$\underset{max}{D(x : \theta_D)} = log(D(x)) - log(1 - D(G(z))) \tag{13}$$

θ_D are the parameters which are updated for the discriminator during the training of the network. $1 - D(G(z))$ is the probability, from the discriminator, a sample is fake. By maximizing $1 - D(G(z))$, the discriminator is incentivized to correctly identify "fake" samples or else it is penalized.

The generator of a GAN takes in a random input of noise from a latent space and generates synthetic data samples for analysis by the discriminator.

$$\underset{min}{G(z : \theta_g)} = log(1 - D(G(z))) \tag{14}$$

Equation (14) represents the original loss function for the generator. The goal is to minimize the log-likelihood loss on the fake images. θ_g is the notation for the parameters of the generator updated during training. The paper later proposes a slight change in the generator's architecture to help overcome saturation occurring during GAN training. The revised loss function,

$$\underset{max}{G(z : \theta_g)} = log(D(G(z))) \tag{15}$$

as stated by Goodfellow et al., helped provide stronger gradients for the earlier stages of backpropagation in the network's training while keeping the relationship of the discriminator and generator in the overall loss function the same [57].

In 2015, the Deep Convolutional Generative Adversarial Network (DCGAN) was proposed by Radford et al. [1] which used CNNs in its architecture for generating

synthetic image data. The addition of convolutional layers decreased the number of trainable parameters needed in the network as well as added more stability to the network's training.

In 2017, Arjovsky et al. [58] introduced the Wasserstein Generative Adversarial Network (WGAN) which seeks to approximate the Earth Mover (EM) distance, the distance between two different probability distributions, instead of a sample being "real" or "fake." The WGAN uses the divergence between the real dataset's probability distribution and the generated dataset's probability distribution. Currently, more efficient ways of training GANs and methods for validating their augmented samples are being pursued in this relatively new area of research.

2.6.2 Convergence

As a GAN approaches convergence, the synthetic data produced will slowly match the distribution of the original data. At the final convergence point, the discriminator and generator will both be optimal, and the distribution of the real data will exactly equal the distribution of the generated data. GANs exploit measurements in probability distribution divergence equations for generating more realistic data. GANs were originally contrived to use the Kullback-Leibler (KL) divergence. KL divergence

$$KL(\mathbb{P}_r||\mathbb{P}_g) = \int log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x)d\mu(x) \tag{16}$$

is the distance between two probability distributions. As the probability distribution of augmented data points becomes similar to the original dataset, this distance goes to zero.

The Jensen-Shannon (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL\left(p_r || \frac{p_r + p_g}{2}\right) + KL\left(p_g || \frac{p_r + p_g}{2}\right) \tag{17}$$

builds off of the KL divergence by adding a symmetric property to the equation. Just like KL divergence, the JS divergence will go to zero when the generated data's distribution is exactly same as the original data's probability distribution. At this point, the discriminator and generator will both be optimal.

2.6.3 Generative Adversarial Network Training Problems

Two problems which occur during the training of a GAN are vanishing/exploding gradients as well as mode collapse. The vanishing/exploding gradient problem is comparable to the occurrence of the problem in a CNN and follows the same pattern of gradients which are either too small to have an effect on efficiently updating the weights or too big and cause the model to explode and be rendered ineffective. The mode collapse problem leads to lack of diversity in generated data. The discriminator's gradient for similar points will all point in similar directions. Salimans et al. [59] attributed mode collapse to the discriminator's independent analysis of real and fake samples which leads to no coordination for updating the gradients. This further leads to no guidance being provided to the generator when being updated to ensure variety of its outputs. With no direction motivating diversity, the generator shifts its output to a single or handful of synthetic points which the discriminator always labels as "real." The researchers proposed minibatch discrimination to combat this problem. Minibatch discrimination forces the discriminator to look at multiple data samples rather than just one sample independently. The discriminator's goal to classify as "real" or "fake" stays the same but it now uses information from all samples in the same batch and classifies the batch, giving the discriminator the ability to find data

points which are fake but closer to the datasets original distribution. This leads to higher quality synthetic data over time as the generator is now forced to create more realistic data samples for all classes of the dataset.

Originally, GANs were unconditioned and produced data which was unrepresentative of the original dataset's distribution and physical features. When GANs are unconditioned, they randomly try to estimate the distribution of the original dataset while mapping its features to a latent space. Mirza et al. [60] produced research which conditioned a GAN on class labels of the original dataset, providing the architecture for a Conditional Generative Adversarial Network (cGAN). They changed the loss function of the GAN to

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}}(x)[\log D(x|y)] + \mathbb{E}_{z\sim p_z}(z)[log(1 - D(G(z|y)))] \quad (18)$$

where y is the conditional component of the model. The discriminator takes in an input image and a label which is embedded through an embedding layer. The embedding layer vectorizes the label for input into the convolutional layers. The label is then concatenated to the image and critiqued on its level of authenticity by the discriminator. cGANs are still subjected to the problems occurring in GAN training, however, the condition placed on the GAN greatly diminishes the instability issue. While Mirza et al. [60] recommended further research to create more efficient Conditional Generative Adversarial Networks (cGANs), one positive takeaway from their research is the ability to have some choice over the distribution of images produced using a label and to take control of an otherwise random training process of a GAN.

2.6.4 Architecture

The architecture of a GAN is composed of the discriminator and generator which are trained through an iterative process. For creating augmented images, while the

discriminator and generator mimic the general structure of a CNN, there are some unique components of their respective architectures which make them distinctly different and lead to synthetically created data. The generator's architecture is designed to slowly increase the size of its generated images as the input noise vector goes deeper into the architecture, a process defined as "upsampling" or the transpose of the matrix convolutional operation. The generator will produce fake images and these images will then be given to the discriminator. The discriminator will analyze batches of real and fake images separately, having a calculated loss which is the average of the loss from the fake images and the loss from the real images. The weights of the discriminator will update in accordance to its overall loss and the weights of the generator will update according to the loss on the fake images only.

2.6.5 Deep Convolutional Generative Adversarial Network (DCGAN)

Radford et al. [1] proposed replacing the fully connected layers in a GAN with convolutional layers to add stability to the GAN training process, especially for image datasets. The DCGAN they used was an unconditioned GAN utilizing unsupervised learning. The datasets the researchers used to investigate the DCGAN framework were the Large-Scale Scene Understanding (LSUN) dataset [61], and a dataset compiled of human faces.

The discriminator's architecture was guided by the LReLU Activation Function with a slope of 0.2 in all its layers. The final layer of the discriminator was a flatten layer and concluded with a Sigmoid Activation Function. The discriminator seeks to maximize the probability that a training sample came from the real dataset and not the augmented dataset. It will be penalized for misclassifying a sample and uses the same loss function proposed in the original paper for Goodfellow et al [57]. The discriminator will receive batches of real and fake images and the loss is then the

average of the losses from the real and synthetic batches. The discriminator also incorporates batch normalization which normalizes the input at each node to have a zero mean and zero variance [1]. No batch normalization was implemented on the input layer of the discriminator.

The generator's architecture, depicted in Figure 13, was composed of five transpose convolutional layers, starting with a 4x4 image and finishing with a generated 64x64 image. The input for the generator was a noise vector emerging from a latent space expanding 100 dimensions. The hidden layers all utilized the ReLU Activation Function while the output layer used a tanh Activation Function which meant the images had to be scaled between -1 and 1 in the preprocessing steps of the training. No batch normalization was used on the output layer of the generator.

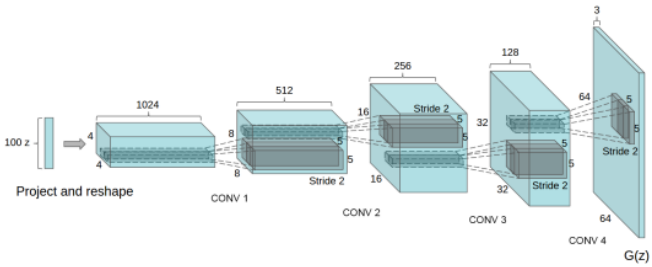


Figure 13: DCGAN Generator (Image from [1])

Their research provided some notable characteristics of GANs. First, GANs have the ability to learn different features of a dataset and can draw information from specific objects. Additionally, generators have the ability to combine different features due to the mapping of features of vectors into a latent space. These conclusions led to the support of the proposal stating GANs learn to exploit a latent space and map features to it.

2.6.6 Wasserstein GAN (WGAN)

The WGAN increased the stability of GAN training by the orientation of its loss function around the difference in probability distributions. The original GAN architecture used KL divergence for reaching convergence whereas the WGAN seeks to approximate the EM distance, also known as the Wasserstein-1 Distance,

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\Big[||x - y||\Big] \tag{19}$$

which is the distance between two different probability distributions where the parameters $\mathbb{P}_r$ and $\mathbb{P}_g$ are the real and fake data distributions, respectively. $\prod(\mathbb{P}_r, \mathbb{P}_g)$ represents all the joint distributions of $\gamma(x, y)$. Arjovsky et al. stated "$\gamma(x, y)$ indicates how much 'mass' must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ in the distribution $\mathbb{P}_g$," [58]. The EM distance is continuous and differentiable which makes it a more efficient loss metric for training GANs rather than configuring GANs to exploit KL and JS divergence. As shown by Figure 14, the EM distance has a gradient at all points in its function.

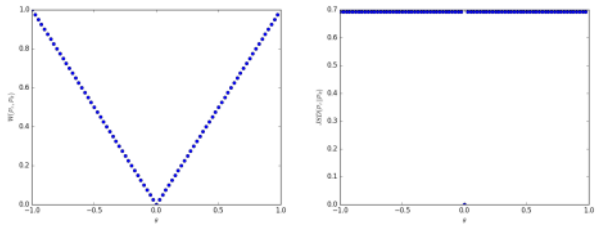


Figure 14: EM Distance vs. JS Divergence: Gradients (Image from [58])

The architecture and training of the WGAN proposed by Arjovsky et al. is summarized in Algorithm 1. One major deviation from the DCGAN is the use of the RMSProp Optimizer instead of the Adam Optimizer. The researchers found one of the main issues causing the instability in the WGAN was using a momentum based

optimizer. The use of the RMSProp was chosen since it works better for nonstationary problems [62] and the discriminator, called the "critic" by the authors, is nonstationary in nature. The *for* loop in the algorithm shows the training of the critic for n generations before the generator is trained, a unique feature of the WGAN. Gradient clipping is used to enforce a Lipschitz constraint which helps control vanishing and exploding gradients occurring during GAN training.

Algorithm 1 Wasserstein GAN Algorithm proposed by Arjovsky et al. [58]. Default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$,$n_{critic}= 5$

Require: α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: w_0 initial critic parameters. θ_0, the initial generator's parameters.
while θ has not converged **do**
  for $t = 0,..., n_{critic}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \Delta_w[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))]$
    $w \leftarrow w + \alpha \cdot \mathrm{RMSProp}(w, g_w)$
    $w \leftarrow \mathrm{clip}w(c, -c)$
  end for
  Sample $\{x^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\Delta_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \mathrm{RMSProp}(\theta, g_\theta)$
end while

2.6.6.1 Lipschitz Constant

To overcome instability issues in training, a Lipschitz Constant is used. For their research, Arjovsky et al. [58] used gradient clipping to enforce the constant. The clipping rate becomes a hyperparameter for the WGAN training process. The Lipschitz Constant creates a bound on the rate of change in the weights of the critic. The authors did state how weight clipping may not be the most optimal way to execute the use of the constant but it worked for this configuration of the WGAN.

2.6.6.2 Performance Experimentation

The architecture of the WGAN is similar to the DCGAN but is changed in a few unique ways. The discriminator is renamed to "critic" by the authors since it does not utilize a Sigmoid Activation Function in its output layer, replacing it with a LReLU Activation Function with a slope of 0.2. This means the critic is now "critiquing" how realistic a sample is rather than labeling a sample as "fake" or "real." The loss function for the critic is changed to

$$\underset{max}{L(D, g_\theta)} = \mathbb{E}_{\sim \mathbb{P}_r}[logD(x)] + \mathbb{E}_{\sim \mathbb{P}_\theta}[log(1 - D(z))] \tag{20}$$

For testing the stability of the WGAN, Arjovsky et al. [58] selected the LSUN dataset used in the original DCGAN implementation. The results of their GAN showed saturation when using the Adam optimizer, therefore they changed the optimizer to the RMSProp optimizer with a learning rate of 0.00005. For their clipping parameter, they used a value of 0.01. Unlike the DCGAN, the critic is updated five times before the generator is updated. To test the power of the WGAN variant, the researchers used different GAN architectures to investigate the robustness and stability of the WGAN framework. One architecture used a convolutional DCGAN generator, a second used a convolutional DCGAN generator but batch normalization was removed and the number of filters was kept constant, and the third architecture was a four layer ReLU-MLP which had 512 hidden units.

The researchers' experiment showed how as the EM distance became smaller over generator iterations, the quality of the images improved. Both of the architectures structured after the DCGAN were stable during training while the ReLU-MLP was unstable due to the high learning rates. The two main takeaways from their research were the WGAN was much more stable than prior GAN architectures, and the loss

metric of the GAN has more meaning since a smaller EM distance is correlated to the generator and critic moving towards convergence. Another important finding by the researchers was at no point in their research did mode collapse occur when using the WGAN architecture.

2.7 Related Research

2.7.1 Model Generalizability

CNNs have shown evidence of being superior classifiers for image data classification, especially in processing aerial image data. Sheppard and Rahnemoonfar [63] were able to classify Unmanned Aerial Vehicle (UAV) data imagery of agriculture with an accuracy of 93.6%. Their network architecture consisted of two convolutional layers each followed by batch normalization and a max pooling layer which reduced the image features by 50% for each pass. Each layer had a ReLU Activation Function as well and was followed by a 50% dropout rate. The researchers provided proof of model detection generalizability by obtaining and presenting insights on a correct prediction from their CNN for "round" buildings which were never seen in training. All the buildings in the training of the CNN were square in shape. Their research provides evidence of attainable increases in generalizability and learning ability for CNNs to classify images in a class distinctly physically different from their original class used for training the CNN.

With an imbalanced dataset, having a robust model that is generalizable to huge amounts of similar data with small deviations is difficult to achieve. Salehinejad et al. [64] examined medical image datasets and showed how balancing image datasets will increase the generalizability of the classification models which inspect the image samples. They used a DCGAN to generate new images of medical x-rays portraying different diseases affecting the lungs. The researchers' classification model was a

CNN similar in architecture to AlexNet, using ReLU Activation Functions with L2 normalization in each of their convolutional layers and following those layers with max pooling layers. For model validation, k-Folds cross validation was used with 10 folds. The first trial of their experiment trained their neural network with the real imbalanced dataset. This model had an accuracy of 70.87% with a standard deviation of 0.47%. The second trial trained the model on the real training set with majority of the images removed to achieve balance. This significantly decreased the accuracy to 58.90% with a standard deviation of 0.48%. The last trial used the real imbalanced dataset augmented with the synthetic data. This produced enormous improvements in the model's accuracy, obtaining an accuracy of 92.10% with a standard deviation 0.41%. Not only did the model's accuracy increase, but the standard deviation decreased. The research showed how huge improvements in CNN generalizability and accuracy can be obtained from image datasets augmented with synthetic images generated by a GAN.

One weakness in the researchers' image generation was the technique they utilized for assessing the quality of generated images and accepting a sample for augmenting the dataset. The experimenters had a radiologist look at the generated images and decide whether an image looked acceptable dependant upon the lung condition it was supposed to portray. One downfall to using this qualitative image assessment technique is if the images were dissimilar quantitatively to the original dataset but were approved for use in the training set, this could have skewed the authors' work and negatively impacted the performance of their detection model.

Wang et al. [25] conducted a case study to understand how a CNN model's accuracy was affected when the use of synthetic data generated from a GAN was used to increase the size of the training dataset. They compared the effectiveness of changes in the accuracy of one-dimensional and two-dimensional CNNs to a SVM, a

MLP, and a k-nearest neighbor algorithm to understand how ML models can benefit from GAN augmentation as well to understand if the performances of the CNNs were more sensitive to changes in the dataset. The motivation for their research was to increase the size of a small and imbalanced photovoltaic power forecasting weather dataset.

To augment their dataset, Wang et al. utilized a Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP). The model metrics used to investigate changes in the models' performance were Overall Accuracy (OA), which was the overall accuracy of the model, Product Accuracy (PA). which is the same measurement as recall rate, and User Accuracy (UA), which is also known as precision rate. Each of the five different models were trained twice. The first training iteration for each model used the original dataset with no augmented data. The second training iteration used the original set augmented with synthetic data from the WGAN-GP. The two-dimensional CNN achieved the highest accuracy compared to all of the other models on the original dataset, obtaining an OA of 76.90%. After the two-dimensional CNN was trained with the augmented datasets, its accuracy increased to 89.00%. In addition, all of the UA and PA rates either drastically increased or stayed the same if they were at or close to 100.00%, further showing a GAN's ability to improve the performance of a CNN classifier. There was only one decrease in all of the model's rates which was the UA rate for one class of data. All of the other models followed a similar pattern of performance increases when trained with the augmented dataset.

2.7.2 Aerial Image Dataset Augmentation

Bang et al. [5] implemented three different data augmentation techniques for images acquired from moving platforms depicting construction sites. Their goal was to increase their model's object detection performance for different classes of objects

found in construction sites such as concrete trucks, dump trucks, workers, and crane trucks. The object detection model was a R-CNN. The original dataset contained only 655 images and had imbalances between the different classes of objects. The number of objects in the dataset were calculated and their probability distributions were found to determine how many images of each object should be added back into the sample frames for balancing the dataset.

The three methods Bang et al.[5] used for augmenting their dataset were called the removing-and-inpainting technique, cut-and-paste technique, and a combination of image transformation techniques. The removing-and-inpainting technique consisted of two different modules and was used as preprocessing for the cut-and-paste technique. The first module was named "object-removing" and would randomly select objects in an image and convert them to white mask. The second module was called "image inpainting" and used a GAN to create new pixels in the white masked areas of the cropped out objects to blend in the blank space with the pixels that were similar in color to the background surrounding the space. Once the background information was filled, the cut-and-paste technique was used to overcome class imbalances in the dataset and add under represented objects to the images. Finally, intensity-, blur-, and scale-variations were used to see how these augmentation techniques on aerial image datasets affected model performance.

Five different trials were conducted on the model to understand how its performance would be impacted. One trial consisted of no augmentation techniques (baseline trial), three trials were implemented using solely one of the three data augmentation methods, and one trial consisted of all three of the data augmentations combined. The results showed how using the three techniques in conjunction with each other produced the greatest increases in model performance. The model's recall rate improved from 35.96% to 60.22% , the precision rate changed from 45.62% to

57.13%, and the F-measure rate increased from 40.22% to 58.63%. Bang et al. concluded how the use of different objects in different backgrounds had the biggest affect on precision rate improvements and the image transformation techniques on the data had the largest impact on improvements in the recall rate.

The researchers had some flaws in their analysis and outlined some of the problems which may have negatively impacted the training of their R-CNN. One flaw involved images procuring dramatic problems in the scaling of the objects.

Figure 15: Object Scale Imbalance (Image from [5])

As shown by Figure 15, the scaling of the dump truck compared to the crane and worker is distorted. While objects closer in images tend to appear bigger, the background of the image is not consistent in scale to where the dump truck sits. The researchers did address a problem of their analysis where some of the objects placed in the images were put in implausible areas which may have led to degradation of the model's performance. An additional flaw, as outlined by Bang et al, was the weather conditions and the terrain of the images were not taken into account for the data augmentation. For recreating the objects, the color and shape of the objects was not changed, therefore, decreasing the diversity of the construction assets used to train the model on to detect. Diversity of images is important for strengthening a model's generalization on variations in datasets. For the use of a GAN, it would have been beneficial to investigate the recreation of new objects in the construction sites

as well, rather than just filling blank sections of the images with pixels mimicking the background information, for showing the model new samples and understanding which factors effect the model's generalizability the most.

Shen et al. [65] used a similar technique as the Bang et al. technique which segmented the important features of their dataset away from the original frames consisting of aerial images of vehicles to synthesize new features with a GAN. One of the datasets they concentrated on was the Munich Dataset, a dataset consisting of twenty aerial images taken over the city of Munich, Germany. The second dataset used for model evaluation was created by the authors and was a collection of 615 aerial pictures of vehicles. They used a "lightweight" CNN since it used less parameters and had a faster convergence speed in training. For the GAN, they created a new GAN variant, called a Multi-conditioned Constrained Generative Adversarial Network (MC-GAN), conditioned on various attributes of the dataset. The GAN combined pix2pix image translation with a cGAN to achieve their defined goal "to generate realistic vehicles but more importantly to ensure that the generated vehicles are integrated into the background information." The first process segmented apart the object of interest from the inside of a frame. Then, they added noise to where they made the segmentation cut in the original image. The GAN had two discriminators: one which decided whether the augmented vehicle was "real" or "fake" and another designed to assess the background information of the generated sample and how well it blended to the original images background.

For evaluating the strengths of their dataset augmentation, the authors looked at the recall, precision, F1-score, and Mean Average Precision (mAP). Their research showed how high percentage values can be obtained for these metrics with small training sets consisting of images acquired by an aerial vehicle further augmented with a GAN. They compared their results to other detection models to assess the efficiency

of a model smaller in size. Their model did much better than many of the other detection models (VGG16, ResNet50, DenseNet-121, 1.0 MobileNet, ShuffleNet-v2), achieving the highest accuracy of 86.90%. Their methodology also showed the importance of directing the GAN's augmentation efforts towards the object of interest in the image rather then augmentation of the entire frame. Their research also supports the use of object detection models for aerial images with less parameters thus smaller computational size since they have the ability to achieve higher rates of accuracy, precision,and recall rates when trained with datasets augmented with data generated by a GAN while still being deployable on a smaller aircraft. One problem, as outlined by the researchers, was some of the generated images could be seen to have variable amounts of noise and looked distorted in nature. These issues are due to the instability in the training of a GAN.

Finally, Neagoe et al. [66] used an unconditional GAN for augmenting hyperspectral UAV image datasets. Their dataset consisted of hyperspectral aerial images size 610x340. The classification network created was a Deep Convolutional Neural Network (DCNN) which consisted of five convolutional layers and five Gated Recurrent Units (GRU) layers. Through hyperparameter tuning, the researchers discovered their model performing better when using LReLU Activation Functions in its hidden layers instead of ReLU Activation Functions. Neagoe et al. followed a similar methodology of initially training without augmenting the dataset and then retraining the model after dataset augmentation. With the original dataset and validation of the model, the accuracy achieved was 92.94%. After using an unconditional GAN to generate new samples and augment the existing dataset, the model's accuracy increased to 95.32%. In addition, the time needed to train the model decreased from 41.5 minutes to 32.4 minutes with the augmented dataset. For aerial image datasets, GANs show potential for increasing model performance, decreasing time of training,

and decreasing the time and cost for collecting the data, designating their use as a possible premier choice for dataset augmentation.

III. Methodology

Preamble

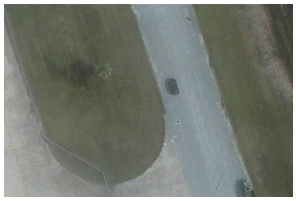
This chapter details the methodology used throughout our investigation. Broadly outlined, we performed data wrangling on the original dataframes to transform them into a usable form by our models. We extracted the salient objects from each frame and compiled them together to train our Generative Adversarial Networks (GANs). We used the GANs to create synthetic images of the dataset's classes and then added these generated images back into the frames. Finally, we trained You Only Look Once Version 4 - Tiny (YOLOv4-Tiny) on both unaugmented and augmented training sets.

3.1 Programming Platform

We used Python 3 for the Generative Adversarial Network (GAN) and object detection models' code. PyTorch was the primary machine learning Python library we used for GAN implementation. YOLOv4-Tiny was executed using Darknet coded with Python 3.

3.2 Datasets

There were two image datasets used in our research. Each dataset was framed as a parent dataset and an image inside a parent dataset was the child image for that dataset. Dataset 1 contained images of vehicles driving along a road captured at an overhead angle.



(a) Example 1

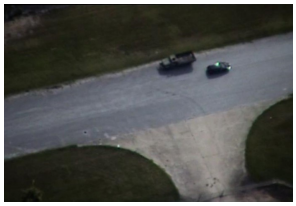
(b) Example 2

(c) Example 3

(d) Example 4

Figure 16: Examples of Dataset 1 Images

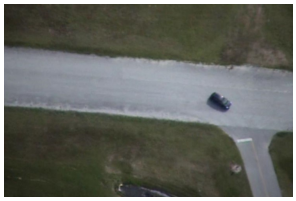
Dataset 2 contained images of vehicles driving along a road captured at a sideways angle. We also created a third dataset. Dataset 3 contained all of the images from Dataset 1 and Dataset 2.



(a) Example 1

(b) Example 2

(c) Example 3



(d) Example 4

Figure 17: Examples of Dataset 2 Images

Previous image augmentation attempts were made on the images. First, each image was positioned at the middle of a black background and sized to 1200x1600.

Some effects added to the original image as they were placed on the black backdrop included image rotation, flipping, darkening, brightening, and shifting.

The image folders contained three different classes of objects : "van", "truck" , and "car." Each image either contained one of the three classes or a random combination of the three objects. The image files also were labeled with the bounding box coordinates and object labels.

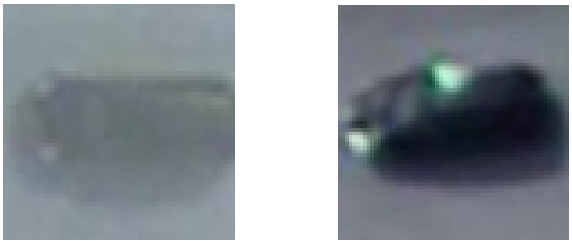
Table 1: Object Count by Class for Each Dataset

Dataset	**Car**	**Van**	**Truck**
1	90	180	101
2	160	64	167
Total	**250**	**244**	**268**

Objects were extracted from the images and visually inspected as well to see if their labeling was accurate to the object's physical features. Due to the quality of the images as well as the height of the moving platform at the time of the images, there were instances where it was difficult for us to visually clarify if a labeled object resembled its given class. Some objects were distorted due to the previous augmentation applications while others were partially cut off if the object was a vehicle entering or leaving the frame at the moment it was captured on camera.

3.3 Datasets' Objects

There were two different instances of the van throughout the datasets. Effects of augmentation and picture purity which led to reduced quality in the child image can be seen in Figure 18a.



(a) Dataset 1 (b) Dataset 2

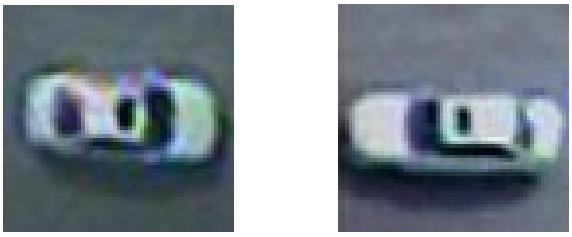
Figure 18: Examples of the Van Class

The second object class was a truck. There was never any frame containing more than one truck.

(a) Dataset 1 (b) Dataset 2

Figure 19: Examples of the Truck Class

The third and final object in the dataset was a white car. Similar to the vans, the car instances differed from each dataset due to the altitude of the aircraft at the moment the image was taken, the lighting during the time the vehicle was captured by the camera, and the data augmentation technique placed upon the image.

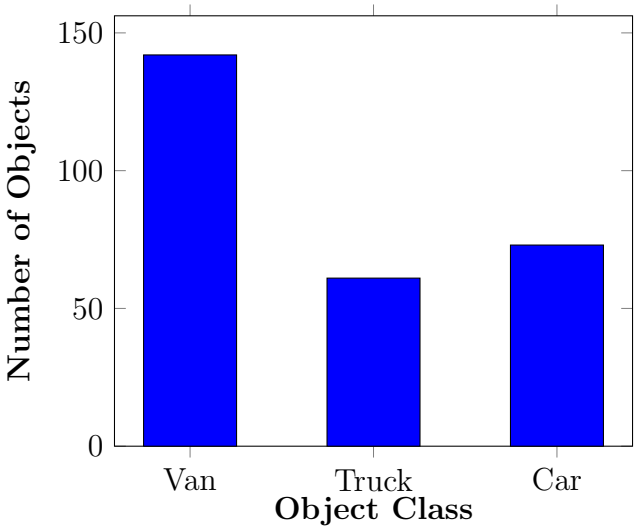


(a) Overhead (b) Sideways

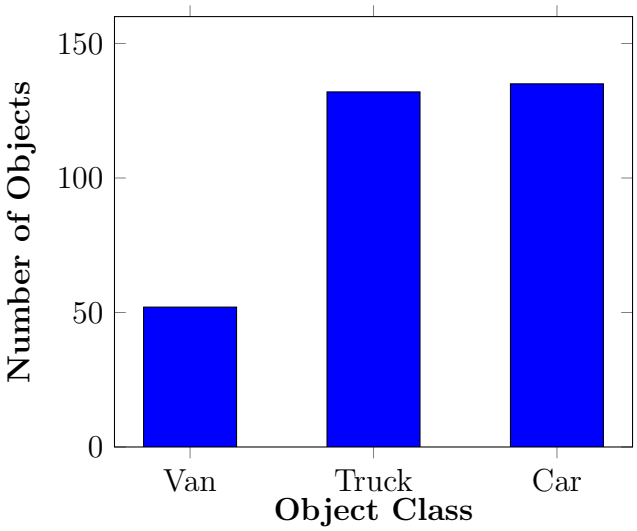
Figure 20: Examples of the Car Class

3.3.1 Splitting the Data for Training and Testing

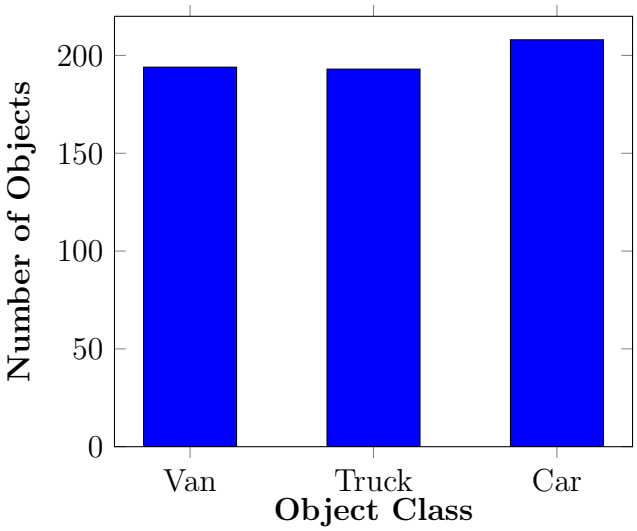
We used an 80/20 ratio for splitting the dataset. The two datasets were first combined together into one inclusive collection of images. 20% of the images from this collection were then taken as the testing dataset for model training. The images were inspected and we used stratification to create the testing set to ensure it closely resembled the distribution of objects and types of images in the training set. For example, Dataset 2 had a collection of images showing a solo white car with a few instances of a truck apparent in the image. There was a total of 90 frames gathered of these instances, and using the 80/20 split, we moved 72 frames to the training set and 18 images to the testing set. The same process of splitting was used for each distinct collection of similar instances.



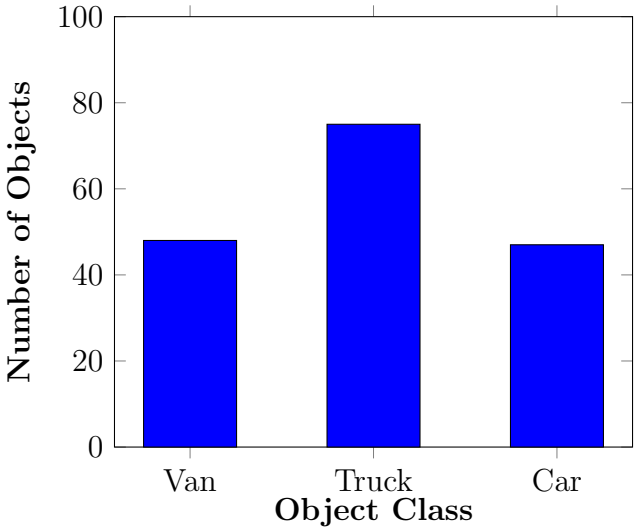
The above bar chart outlines the distribution of objects by class for Dataset 1. There were 142 vans, 61 trucks, and 73 cars. Vans caused the greatest imbalance in the dataset, accounting for 51.45% of the objects captured by the aerial moving platform.



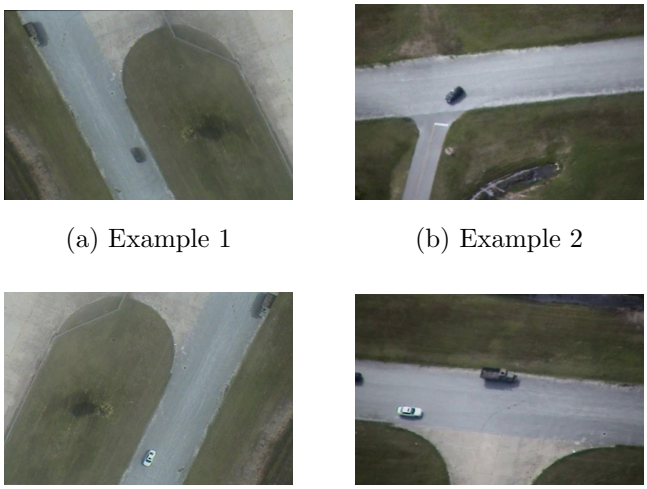
The bar chart above depicts how Dataset 2 was balanced between the number of cars and trucks it contained, however, vans were drastically underrepresented in this dataset. Vans only accounted for 16.39% of the total number of objects in Dataset 2.



As outlined by the chart above, the initial combined dataset was balanced in regards to objects by class, but was still imbalanced by its lack of diversity and size. There were 194 vans, 193 trucks, and 208 cars. Cars had the highest distribution across the dataset, representing 34.96% of all the object instances.



For the test set's distribution of objects, shown in the above bar chart, there were 48 van, 75 truck, and 47 car instances. Trucks were distributed the highest in the test set with occurrences of 44.11% in the test set images. The proportion of vans in the test set was 28.24% and cars were distributed with a proportion of 27.65%. Examples of the test set images can be observed in Figure 21.



(a) Example 1

(b) Example 2

(c) Example 3

(d) Example 4

Figure 21: Examples of Test Set Images

3.4 YOLOv4-Tiny Data Wrangling

We used the steps outlined in Algorithm 2 to ensure the images could be properly read by the object detection model.

Algorithm 2 YOLOv4-Tiny Image Preprocessing Steps

Input: 1200 x 1600 image
Output: Cropped image with a text file containing its annotations.
1: Crop image out from the black background. ▷ Save to a new file
2: Level any images that were previously rotated by data augmentation.
3: Label the objects in the image with their bounding box coordinates. ▷ LabelImg
4: Save the cropped image with its annotations.

3.4.1 Image Cropping

Each image in the original dataset had been inserted onto a 1200x1600 black image. Some of the images were rotated while others were placed undisturbed in the middle of the black image. This technique was previously utilized on the datasets in an attempt the increase the diversity of the datasets' images.

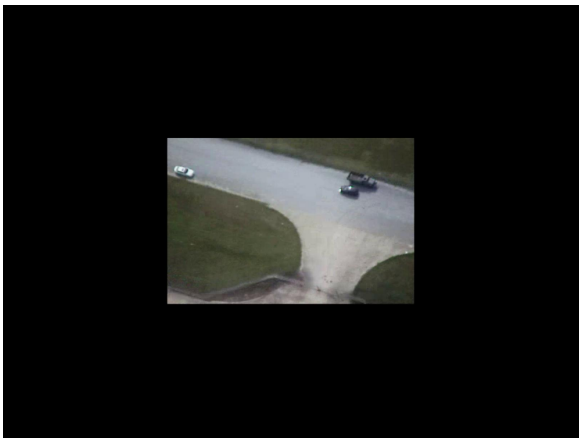


Figure 22: Example of an Original Image

To reduce the size of the images, we hand cropped each image from the black background. All rotated images were reset back to their original orientation to standardize the dataset. Due to the hand-cropping we used on the images, they were

different from each other in their size. This was not a concern since YOLOv4-Tiny does not require a fixed image size for its input. We then relabeled the images based off of the original annotations with the required format for the YOLOv4-Tiny Model.

Figure 23: Example of a Cropped Original Image

3.4.2 Labels

Each image had an associated text file with the object's labels and bounding box coordinates but the coordinates were not in the correct format for the YOLOv4-Tiny Object Detection Model. The original labels were located in a text file with a name matching its associated .jpg image.

Table 2: Original Annotation Format for the Datasets

Label	**Padding**	x_{min}	y_{min}	x_{max}	y_{max}	**Padding**
van	000	497.75	745.62	552.44	795.66	0000000
truck	000	1098.96	351.42	1168.68	415.53	0000000

We used the Python Application LabelImg [67] for relabeling the images since the application has the capability to create bounding boxes around objects of interest in an image as well as add the class label of the object contained within the bounding box in the required YOLOv4-Tiny format.

Table 3: YOLOv4-Tiny Annotation Format for the Dataset

Label	x_{center}	y_{center}	**Width**	**Height**
0	0.126202	0.889423	0.98558	0.091346
1	0.955529	0.33654	0.069712	0.062500

Each coordinate represents a specific measurement of the bounding box surrounding the feature of interest, as shown by Equation (21). One benefit for using this labeling arrangement is the annotation file does not have to be changed if the dimensions of an image are scaled after leveling [68].

$$\textbf{Bounding Box Format} = \left(\frac{x_{center}}{\text{Image Width}}, \frac{y_{center}}{\text{Image Height}}, \frac{\text{Box Width}}{\text{Image Width}}, \frac{\text{Box Height}}{\text{Image Height}} \right) \tag{21}$$

3.5 YOLOv4-Tiny Model Training: No Augmentation

We trained YOLOv4-Tiny first on the three different parent datasets with no augmentation. Due to the small sizes of the datasets, four learning rates were used to investigate the effects on Mean Average Precision (mAP) and Complete Intersection over Union (CIoU) loss, as well as on other quality metrics. The learning rates we used for training the model on the original data and the GAN augmented datasets were 0.00261, 0.001, 0.0001, and 0.00001. The authors of You Only Look Once Version 4 (YOLOv4) stated how their best results had occured with the learning rate of .00261 [23], therefore, we used this learning rate as the baseline for our evaluation.

The configurations of the model depended on the number of classes in the training and test sets. The filters for each of the convolutional layer connected to a YOLO layer had to have an output size of 24 when training the model on three different classes.

The learning rate was scaled by 0.1 at iterations 4,800 and 5,400. The training lasted for 6,000 iterations, with the final results gathered at the last epoch. The values for the preliminary hyperparameters can be found in Appendix A Table 33.

Once the model was trained, several metrics were compiled to assess the model's ability to learn from the training set and make accurate and precise predictions. The True Positive (TP) metric represents the number of times the model was able to detect the original ground truth box with an Intersection over Union (IoU) ≥ 0.5. The False Positive (FP) metric is an assessment for the number of times an object is detected where no object exists or when there the detection for an object occurs at the wrong location of the frame. The False Negative (FN) metric is a measurement for when the model fails in detecting any ground truth bound box.

The precision metric for each class of the model, as well as the overall model, was calculated in the model's output

$$Precision = \frac{\sum_{n=1}^{C} TP_n}{\sum_{n=1}^{C} TP_n + \sum_{n=1}^{N-C} FP_n} \tag{22}$$

where N is the total number deductions, and C is the total number of correct n detections. Precision provided insight on the percentage of times the YOLOv4-Tiny Model correctly made positive predictions. Recall is a measurement outlining the percentage of time the YOLOv4-Tiny Model was able to find the ground-truth bounding boxes in the dataset

$$Recall = \frac{\sum_{n=1}^{C} TP_n}{\sum_{n=1}^{C} TP_n + \sum_{n=1}^{G-C} FN_n} \tag{23}$$

where G is the number of original ground-truth bounding boxes present in the dataset.

An additional metric calculated by the YOLOv4-Tiny Model is the $F_1 - Score$

$$F_1 Score = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{24}$$

which takes the harmonic mean between precision and recall.

One of the main determinants of confirming success in training the YOLOv4-Tiny Object Detection Model is the mean average precision at IOU threshold .5 (mAP@.50) metric. The Average Precision (AP) of the model is calculated with an N-point interpolation value of 101 [68]

$$AP = \frac{1}{N} \sum_{n=1}^{N} Pr_{interp}(R_r(n)) \tag{25}$$

where $P_{rinterp}(R_r(n))$ is a metric which provides insight in the trade-offs of precision and recall for a given model. mAP is the average precision taken over all of the classes [69, 68, 70]

$$mAP = \frac{1}{X} \sum_{i=1}^{X} AP_i \tag{26}$$

where X is the total number of classes.

Another important metric for the model's training is average IoU. [71][68]. The IoU calculation

$$IoU = J(BB_p, BB_{gt}) = \frac{area(BB_p \cap BB_{gt})}{area(BB_p \cup BB_{gt})} \tag{27}$$

is a measurement for the overlapping area of the predicted and ground truth bounding boxes divide by the area of union between the bounding boxes.

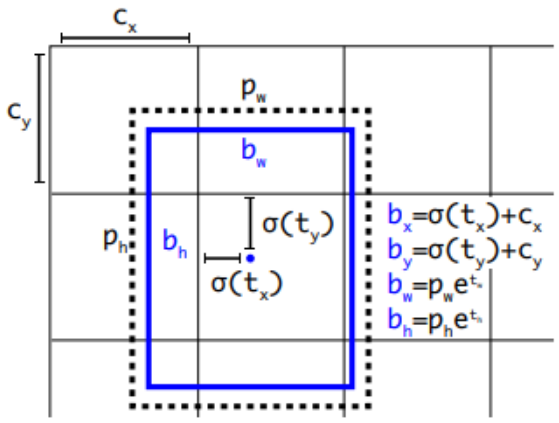


Figure 24: Bound Box Predictions from [72]

Figure 24 shows the parameters for calculating IoU. For each bounding box, the cell is offset by (c_x, c_y) and the network predicts five bounding box coordinates, t_x, t_y, t_w, t_h, and t_o. p_w and p_h are the width and height for the preliminary bounding box. The predictions by YOLOv4-Tiny are b_x, b_y, b_w, and b_h.

IoU is a helpful measure for understanding if the model is correctly localizing an object, however, it fails to provide information during training when bounding boxes do not overlap. This leads to vanishing gradients during the model training process. CIoU loss [73] is a measure used by YOLOv4-Tiny

$$\mathcal{L}_{CIoU} = 1 = IoU\frac{\rho^2(b, b^{gt})}{c^2} + \alpha\upsilon \tag{28}$$

where b is the center point of the predicted box BB_p and b_{gt} is the center point of the target box BB_{gt}. $\rho(b, b^{gt})$ is the distance between the two center points of the bounding boxes. c is the diagonal length of the smallest box contsining the two bounding boxes. υ is a consistency measurement for the aspect ratio

$$\upsilon = \frac{4}{\pi^2}(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h})^2 \tag{29}$$

where the height and width for the predicted box are h and w, and w^{gt} and h^{gt} are the height and width for the ground truth box. Another parameter in the CIoU loss function is the α trade-off parameter

$$\alpha = \frac{\upsilon}{(1 - IOU) + \upsilon} \tag{30}$$

which is formulated to ensure overlapping boxes are prioritized in model training. The use of CIoU loss in object detection models has led to improvements in the accuracy of bounding box predictions by leading to faster model convergence and improved performance [44].

3.6 Generative Adversarial Networks

We trained the four GAN variants, Deep Convolutional Generative Adversarial Network (DCGAN), Conditional Deep Convolutional Generative Adversarial Network (cDCGAN), Wasserstein Generative Adversarial Network (WGAN), and Conditional Wasserstein Generative Adversarial Network (cWGAN), on all three parent datasets. The number of iterations for training was dependent upon the stability of the GAN architecture. Each child object of interest had to be extracted from its respective parent image before being provided to the GAN for learning and generating synthetic replicas. Basic GAN frameworks cannot augment images greater than size 64x64, therefore, the background information of the frames was left untouched. For each parent dataset, objects of the same class were placed into a corresponding folder with their class name. This allowed for the additional use of labels necessary for conditioned GANs. Not all objects extracted from the original images were used for training each GAN.

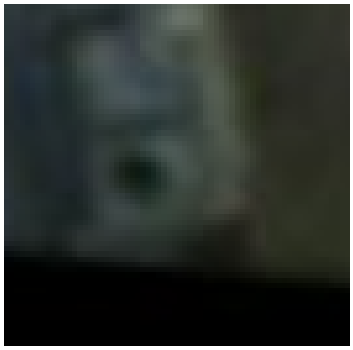


Figure 25: Example of a Sliced Image

Sliced figures, as shown by the example in Figure 25, were not used for GAN training. We omitted any object not fully visible in an image, or too distorted in nature, from the GAN training set to thwart any possibility of low quality or unrealistic recreations.

3.6.1 Generative Adversarial Network Training

Once the image extraction phase was complete, we trained each GAN on the training child images acquired from each parent dataset. For training on the images, each image was normalized using the PyTorch dataloader and transformations. The number of object images saved after each training cessation was constrained by the amount of Graphics Processing Unit (GPU) and Random-access Memory (RAM) memory available. After a GAN was trained, each synthetic image had to be assessed for quality and what object class the image represented. We discarded any image appearing too disfigured or indistinguishable from its class for the ensuing augmentation process.

For the training process of each DCGAN variation, we initialized the weights of the discriminator and generator to the recommendation of Radford's research which found the best results in a zero-centered normal distribution of weights with a standard deviation of 0.02 [1]. The discriminator and generator both utilized the Adaptive

Moment Estimation (Adam) optimizer with the author's recommended β_1 momentum value of 0.5 and a β_2 default value of 0.999. The learning rate during training was 0.0002 and binary cross-entropy was used for the network's loss function. The discriminator and generator were then trained in tandem over i iterations with the discriminator separately accepting batches of real and fake images. A loss value was then calculated for the discriminator which was the average of the loss on real images and the loss on fake images. The generator would create fake examples from a latent noise vector and based off the loss on fake images from the discriminator. The new samples were then passed to the discriminator for reevaluation and the training would cease once the defined number of epochs was complete.

The WGAN variants had subtle differences in the way the GAN was trained compared to the DCGAN due to how the discriminator, now called the "critic," needed to be updated for five iterations before the generator was retrained. The learning rate for the WGAN was smaller than the DCGAN, decreasing from 0.0002 to 0.00002. The loss for the critic changed to the difference between the expected value of the critic's assessment on the real images and the expected value of the critic's assessment on the fake images.

3.6.2 Deep Convolutional GAN Architecture

The DCGAN architecture closely aligned with the architecture proposed by Radford et al. [1] to ensure stability. For the generator, the input was a random latent noise vector with a fixed value of 100, a batch size of 128, a feature size of 64 to match the 64x64 images to be generated, and the number of Red, Green, Blue (RGB) channels, which for color images is three. The summary of the body for the generator is outlined in Figure 26.

```
Generator(
  (net): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(100, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
      (1): ReLU()
    )
    (1): Sequential(
      (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): ReLU()
    )
    (2): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): ReLU()
    )
    (3): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): ReLU()
    )
    (4): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (5): Tanh()
  )
)
```

Figure 26: PyTorch Summary of the DCGAN Generator

All of the layers besides the last layer were followed by a Rectified Linear Unit (ReLU) Activation Function and the last layer was concluded with a Hyperbolic Tangent (tanh) Activation Function which had an output between -1 and 1, matching the normalization on the images.

The discriminator for the DCGAN had an input of the RGB value of three and a feature dimensions value of 64. The body of the discriminator is outlined by Figure 27.

```
Discriminator(
  (disc): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Sequential(
      (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (3): Sequential(
      (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (4): Sequential(
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (5): Conv2d(512, 1, kernel_size=(4, 4), stride=(2, 2))
    (6): Sigmoid()
  )
)
```

Figure 27: PyTorch Summary of the DCGAN Discriminator

Each of the three inner layers were accompanied with a Leaky Rectified Linear Unit (LReLU) Activation Function using a slope of 0.2 as well as batch normalization. The final layer was followed by a Sigmoid Activation Function for the binary output of fake or real mapped to zero or one.. The discriminator was trained at the same rate as the generator to ensure it did not become more efficient at detecting differences between fake and real images faster than the generator could produce high quality images.

3.6.3 Conditional Deep Convolutional GAN Architecture

The cDCGAN followed the same architecture as the DCGAN with the addition of labels used for conditioning the generator's output. The Appendix shows the example code of the cDCGAN adapted from the DCGAN variant [74].

```
Generator(
  (net): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(200, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
      (1): ReLU()
    )
    (1): Sequential(
      (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): ReLU()
    )
    (2): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): ReLU()
    )
    (3): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): ReLU()
    )
    (4): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (5): Tanh()
  )
  (embed): Embedding(3, 100)
)
```

Figure 28: PyTorch Summary of the cDCGAN Generator

Embedding was used for adding the labels of each class and mapping each classes' label to a fixed vector. The input of the generator was changed to accept the embedded vector for each label which was a two-dimensional vector containing a class label and embedding array. The generator was re-coded to now except two additional inputs, the size of the embedding vector and the number of classes. For training the cDCGAN, the embedding size was selected as 100 to be congruent with the size of the noise vector. The class label of each image generated by the generator was reshaped to four dimensions to match the dimensions of each image required by PyTorch and concatenated to each synthetic image before being inspected by the discriminator.

The discriminator's architecture expands from the DCGAN implementation by now accepting the number of classes. The number of channels for the discriminator increased by one to account for the label of each image. For the embedding to be passed through into the discriminator's layers, it had to be two-dimensional in shape of the class label and the total size of the image. This was then reshaped to a four-dimensional vector containing the label, one, image height, and image width before

concatenation of the label to the image to be properly read by PyTorch.

```
Discriminator(
  (disc): Sequential(
    (0): Conv2d(4, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Sequential(
      (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (3): Sequential(
      (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (4): Sequential(
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (5): Conv2d(512, 1, kernel_size=(4, 4), stride=(2, 2))
    (6): Sigmoid()
  )
  (embed): Embedding(3, 4096)
)
```

Figure 29: PyTorch Summary of the cDCGAN Discriminator

3.6.4 Wasserstein GAN Architecture

The architecture of the WGAN was derived from the DCGAN in Figure 26 and Figure 27 with a few subtle changes in how the training was implemented. Many of the hyperparameter values chosen were recommended by Arjovsky et al. [58] in their original implementation. We chose the size of the noise vector to be the same at 100 and the batch size for the WGAN was 64, cut in half from the DCGAN. For placing a constraint on the updates in the gradients of the critic and enforcing the Lipschitz constant, a clipping constant was enforced. The clipping value we set was the suggested value of 0.01. Another difference between the DCGAN and WGAN architectures, outlined by Algorithm 3, is the critic is now trained for a defined amount of iterations before the generator produces new images and has its weights updated.

Algorithm 3 WGAN: Updating the Critic

1: **for** i in Critic_Update_Iterations **do**
2: Generate synthetic images from random noise with the generator.
3: Get the loss, *real_loss*, on the real images by inspecting a batch of the original images with the critic.
4: Get the loss, *fake_loss*, on the synthetic images by inspecting a batch of the newly generated fake images with the critic.
5: Calculate the critic's loss: -(*real_loss-fake_loss*)
6: Update the critic's weights with the weight clip constraints.
7: **end for**

Since optimizers seek to maximize and the critic wants to minimize its loss function, a negative sign is added to the loss function of the critic. As previously defined, the loss for the critic was changed to the expected value between the critic's analysis of the real images and analysis of the fake images. Five iterations were used for training the critic. At the conclusion of the five iterations, the generator was trained and wanted to minimize the negative mean of the critic's expected value assessment on the fake images. The generator was the same as the DCGAN and did not adopt the use of the LReLU Activation Function.

3.6.5 Conditional Wasserstein GAN Architecture

A similar tactic used for conditioning the generator and discriminator of of the DCGAN in Figure 28 and Figure 29 was used for conditioning the WGAN. For the critic's input layer, two new inputs for the class number and size of the image were added to allow acceptance of an embedded label vector. The number of channels accepted in the input of the critic was also increased by one. The same process of concatenating the label to an image was used for both the critic and the generator. An identical procedure had to be used for the generator to add the size of the embedding vector to the generator's input, allowing for the acceptance of the embedded label vector by the generator. The generator then produced an image with an associated

embedded label vector which was the same dimension of the synthetic image. Appendix A shows the example code of the cWGAN adapted from the WGAN variant [74].

3.7 Dataset Augmentation

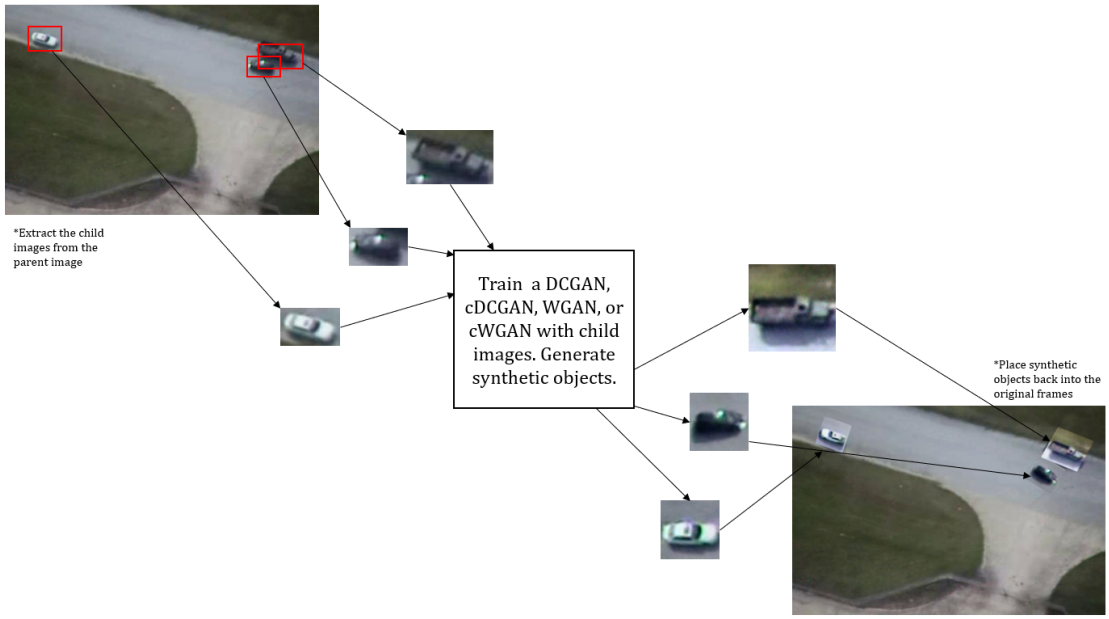


Figure 30: Augmentation Phase Process

In the augmentation phase, shown in Figure 30, we only used synthetic images created from a GAN trained on the parent's child images for augmenting images appended to the respective parent dataset. For implanting the synthetic child images into their related parent images, we utilized the GNU Image Manipulation Program (GIMP) [75]. When adding the child images to their parent frames, we had to resize some of the images from their 64x64 configuration for two reasons. One reason was to cover the original child object in which the augmented child object was replacing. The second reason was to resize the synthetic child images so they matched the size of their replacements to ensure the most realistic training images. Linear interpolation

was used for resizing each of the images. Images were also flipped and rotated to match a similar orientation of the object they were supposed to depict. For the first phase of augmentation, we placed each synthetic child image over the original child image exactly where it originality was located. The dataset was doubled and images were randomly selected for augmentation. The biggest constraint in this stage was the number of available synthetic objects collected from the GAN. If all of the objects in an original image could not be covered with their synthetic counterpart, we did not add the image back to the parent dataset. While the objects obtained from the GANs showed the entire object, GIMP allowed for the objects to be cutoff and replace the objects depicting vehicles driving in and out of the frame.

Once all of the available child images were supplanted in the frames of each parent dataset, we used LabelImg [67] to create new annotations consisting of a bounding box and class label for each of the new augmented objects. Due to possible mislabeling, some of the instances of "car" were changed to "van" from the original dataset due to the drastic similarities between the "car" and "van" objects. We determined the labeled "cars" actually did represent vans and these objects were updated when their augmented replications were placed within the images of the dataset.

Each original parent dataset was used for training the YOLOv4-Tiny Object Detection Model. To further understand the effects of small datasets and assess the effective reinforcement of GAN augmentation, the learning rates of 0.00261, 0.001, 0.0001, and 0.00001 were selected and used to gather evidence and draw comparisons. After training each Parent Training Set on the 4 different learning rates, each augmented training set was then used for training YOLOv4-Tiny. For every sequence, the same test set containing images from the two datasets was used.

IV. Generative Adversarial Networks Training

We trained each GAN variant individually on each of the three datasets. This chapter outlines the details of the GAN training process as well as discusses the quality of the images generated. The biggest constraint occurring during the training of the GANs was the limitations of GPU accessibility and RAM storage.

4.1 Training Times

Table 4: GAN Training Times

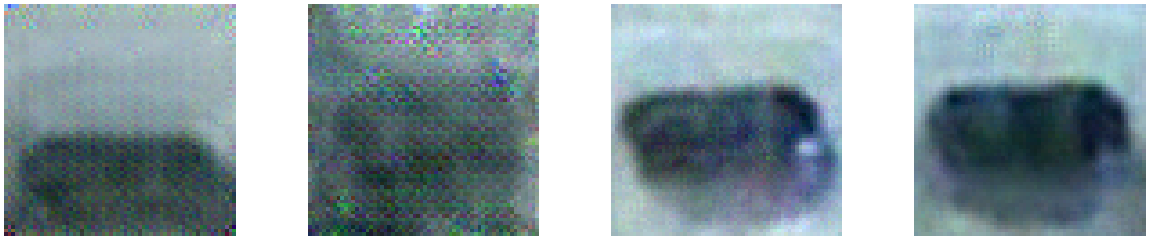
GAN	Dataset	Time (Hours)	Epochs
DCGAN	1	2.405	10,000
DCGAN	2	2.006	8,300
DCGAN	Combined	3.089	10,000
cDCGAN	1	0.968	4,200
cDCGAN	2	2.076	8,300
cDCGAN	Combined	5.388	15,000
WGAN	1	5.736	10,000
WGAN	2	5.840	10,000
WGAN	Combined	10.780	10,000
cWGAN	1	5.715	10,000
cWGAN	2	5.942	10,000
cWGAN	Combined	10.874	10,000

We chose 10,000 iterations as the standard number for training all of the GANs due to software restrictions. The DCGAN training on Dataset 1, cDCGAN training on Dataset 1, and cDCGAN trained on Dataset 2 were unstable when we attempted to train them with 10,000 iterations. The output we obtained when these GANs were trained for 10,000 iterations was either images filled with noise or images displaying indistinguishable objects. In addition, the cDCGAN trained with the combined dataset was producing lower quality results at 10,000 iterations and we found success

when training the cDCGAN with 15,000 iterations. The conditional component stabilized the training with the cDCGAN but needed longer training time to generate quality images for the larger dataset. All of the WGAN variants took much longer to train than the DCGAN variants which we accredited to the use of the Wasserstein loss function in the GAN as well as the way the discriminator (critic) was updated during the training process.

4.2 Dataset 1 Synthetic Images

Overall, we had the most difficulty training a GAN on Dataset 1 due to the smaller number of images for the objects in addition to the quality of the images. The GANs were unsuccessful in learning the truck class due to the limited number of instances available for training from Dataset 1.



(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

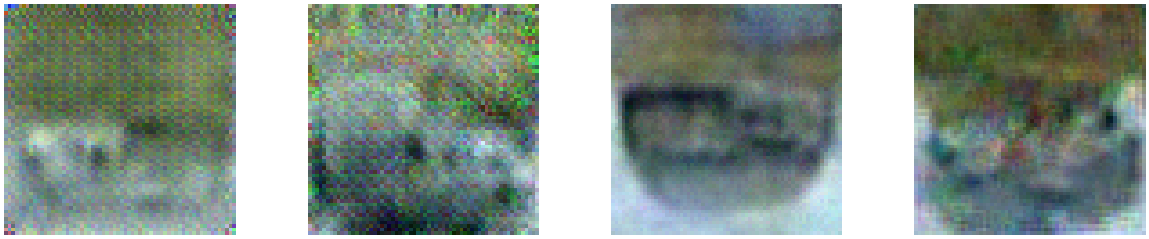
Figure 31: Examples of Synthetic Vans: Dataset 1

While each GAN was able to generate a van, many of the synthetic instances were just black pixels tightly compressed together. Neither of the DCGAN variants could learn the van well, and the generator from the DCGAN simply generated dark shapes to try to trick the discriminator.

Table 5: Percent Change in Vans: Dataset 1

GAN	Generated Instances	Percent Change
DCGAN	0	0.00%
cDCGAN	0	0.00%
WGAN	95	66.90%
cWGAN	10	8.45%

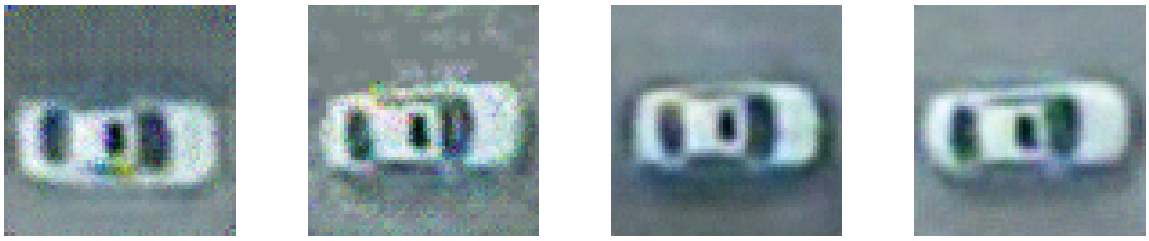
As outlined by Table 5, the WGAN was the most successful variant in producing synthetic vans with a 66.90% change increase in the number of vans in the training set.



(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

Figure 32: Examples of Synthetic Trucks: Dataset 1

The only GAN able to learn from the original images and generate new trucks was the WGAN. As shown in Figure 32, the other GANs collapsed during training and provided indistinguishable depictions of the truck class. While the WGAN produced instances of the truck which were to some degree the desired output, no instance was deemed to be of acceptable quality to add to the training set. One factor attributing to this problem was the few examples of the truck objects for the GANs to use for learning. The GANs simply created an image of noise which could pass the discriminator's (or critic's) real image analysis.



(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

Figure 33: Examples of Synthetic Cars: Dataset 1

All of the GANs were able to produce high quality images of the car. Comparing the images from the DCGAN variants to the WGAN variants, traces of RGB noise are visible in the DCGAN variant images. This is due to the DCGAN architecture and loss function being less stable than the architecture of the WGAN. As outlined by Table 7, the cDCGAN was the most successful in generating new cars producing a 97.26% change in the training set.

Table 6: Percent Change in Cars: Dataset 1

GAN	**Generated Instances**	**Percent Change**
DCGAN	55	75.34%
cDCGAN	71	97.26%
WGAN	54	73.97%
cWGAN	55	75.34%

The images below show the new parent images augmented with the generated child images.

(a) DCGAN

(b) cDCGAN

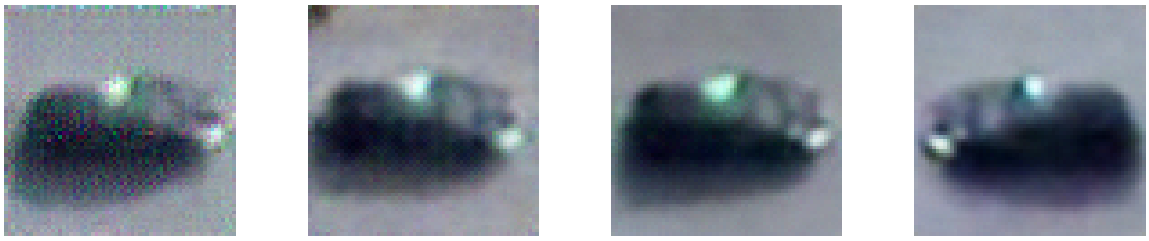
(c) WGAN

(d) cWGAN

Figure 34: Examples of Augmented Parent Images: Dataset 1

4.3 Dataset 2 Synthetic Images

Dataset 2 supplied a much better training set for the GANs than Dataset 1. This was attributed to Dataset 2 containing more objects which were viable options for training each GAN. The training instances of the van class increased substantially due to the possible mislabeling in Dataset 2. The significant increase in vans successfully balanced the original dataset, an important prerequisite for training an unbiased and stabilized detection model.



(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

Figure 35: Examples of Synthetic Vans: Dataset 2

All four of the GAN variants produced realistic generations of the van. The only GAN where traces of RGB noise could be seen was the DCGAN. Additionally, the GANs were able to learn the reflection of light on that van occurring at the time the van was captured on film.

Table 7: Percent Change in Vans: Dataset 2

GAN	**Generated Instances**	**Percent Change**
DCGAN	141	271.15%
cDCGAN	135	259.62%
WGAN	138	265.38%
cWGAN	142	273.08%

The cWGAN had the largest increase in number of augmented vans supporting a 273.08% change in the training set, however, all of the GAN variants were able to produce enormous percent changes in the van class.

(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

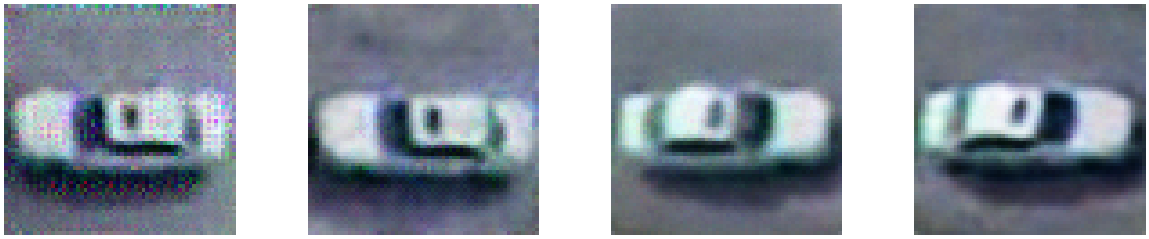
Figure 36: Examples of Synthetic Trucks: Dataset 2

The GANs trained on Dataset 2 did a much better job of recreating the truck. Not only was the vehicle able to be recreated, each GAN was able to learn and generate the shadow produced by the truck in the original dataset. The creation of synthetic trucks followed a similar pattern as the generation of the synthetic vans where the DCGAN showed small signs of instability, creating small markings of RGB noise.

Table 8: Percent Change in Trucks: Dataset 2

GAN	**Generated Instances**	**Percent Change**
DCGAN	132	100.00%
cDCGAN	132	100.00%
WGAN	131	99.24%
cWGAN	132	100.00%

As outlined in Table 8, the DCGAN, cDCGAN, and cWGAN all produced a 100.00% change for the truck class in the training set which was a dramatic improvement from Dataset 1



(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

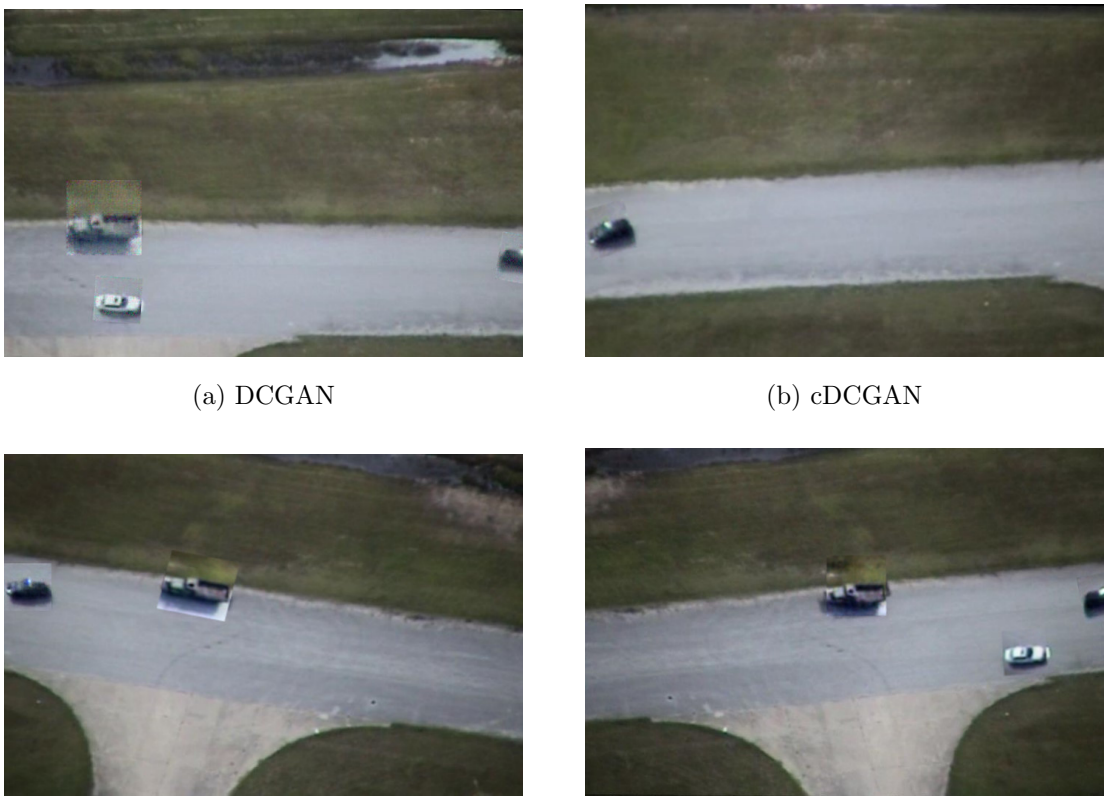
Figure 37: Examples of Synthetic Cars: Dataset 2

For the car class, all of the GANs were able to successfully generate synthetic images. The only trace of RGB noise again occured from the DCGAN. For Dataset 2, the car class had the lowest generation of new images for all GANs.

Table 9: Percent Change in Cars: Dataset 2

GAN	Generated Instances	Percent Change
DCGAN	58	43.94%
cDCGAN	3	2.27%
WGAN	61	46.21%
cWGAN	59	44.70%

All of the GAN variants, as shown in Table 9, were capable of producing an adequate amount of new cars except the cDCGAN which performed poorly in supplying the dataset with synthetic car instances and only produced a 2.27% change in the training set.



(a) DCGAN

(b) cDCGAN

(c) WGAN

(d) cWGAN

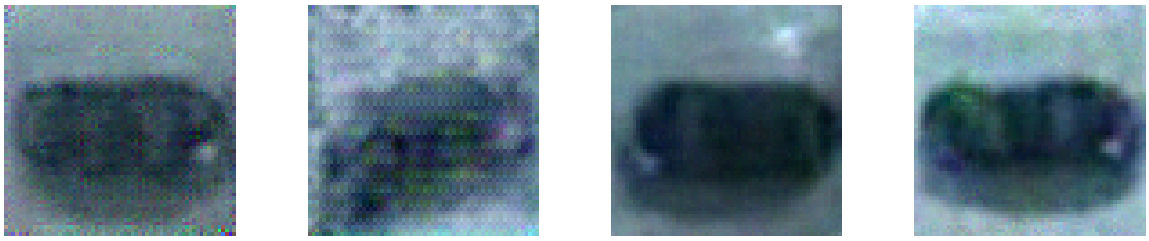
Figure 38: Examples of Augmented Parent Images: Dataset 2

Figure 38 shows different variations of images augmented from the objects created

by GANs. Unlike Dataset 1, trucks could now be added to the original parent images.

4.4 Combined Dataset Synthetic Images

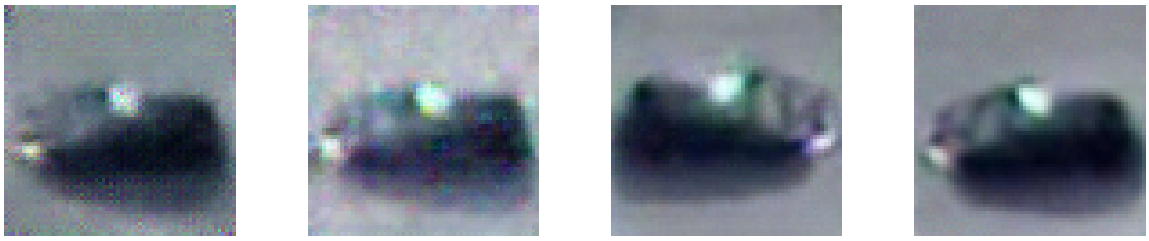
The GANs trained on the combination of the two datasets generated the highest quality images visually compared to datasets 1 and 2. This factor is related to the increase in images provided for the GANs to learn from and map the features in the images to a vector space. Out of all four of the GANs, the cDCGAN produced the lowest quality images. Even though adding the labels helped stabilize the DCGAN architecture, the cDCGAN relied more on producing the right class for each label and did not accurately learn to output an acceptable resolution quality of the images present in the original objects. While generation for Dataset 1 showed weaknesses in creating new trucks, Dataset 2 was capable of producing the truck, and using them in tandem helped overcome the difficulty in generating the complex object and created the visually best instances of the truck class.



(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

Figure 39: Examples of Synthetic Vans: Combined Dataset

The only GAN having trouble with generating the first van variation was the cDCGAN. Unlike prior GANs, all three of the other GANs producing recognizable vans also had the headlight and shading features originally attached to the van.



(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

Figure 40: Examples of Synthetic Vans: Combined Dataset

For the second style of van, all of the GANs were successful in its recreation. The GAN with the weakest regeneration of the van was the cDCGAN and the GAN producing the strongest recreation of the van was the cWGAN. RGB noise was visible in both of the DCGAN variants.

Table 10: Percent Change in Vans: Combined Dataset

GAN	**Generated Instances**	**Percent Change**
DCGAN	245	126.29%
cDCGAN	162	83.51%
WGAN	156	80.41%
cWGAN	97	50.00%

Table 10 shows how all of the GAN variants were capable of producing large amounts of synthetic van instances with the DCGAN creating the most amount of new images, leading to the highest percent change of 126.29%.

(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

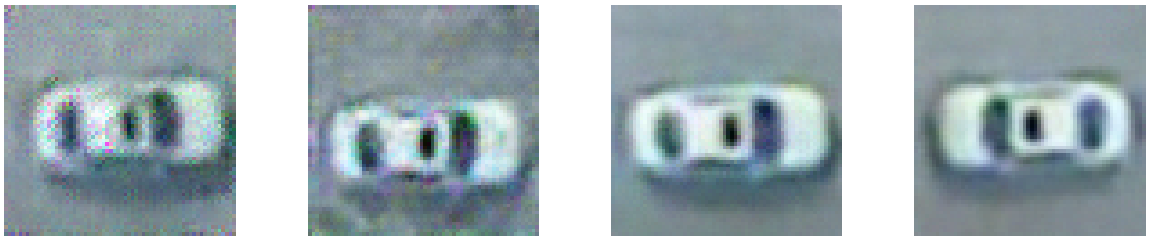
Figure 41: Examples of Synthetic Trucks: Combined Dataset

The RGB noise is more noticeable in the instances of the trucks for the GANs adopting the DCGAN architecture while both of the WGAN architectures were able to produce extremely clear and well-defined trucks.

Table 11: Percent Change in Trucks: Combined Dataset

GAN	**Generated Instances**	**Percent Change**
DCGAN	115	43.56%
cDCGAN	106	40.15%
WGAN	103	39.02%
cWGAN	37	12.29%

As shown in Table 11, the cWGAN produced the lowest amount of new truck instances with a 12.29% change in the dataset which can partly be attributed to lack of memory during the training process.



(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

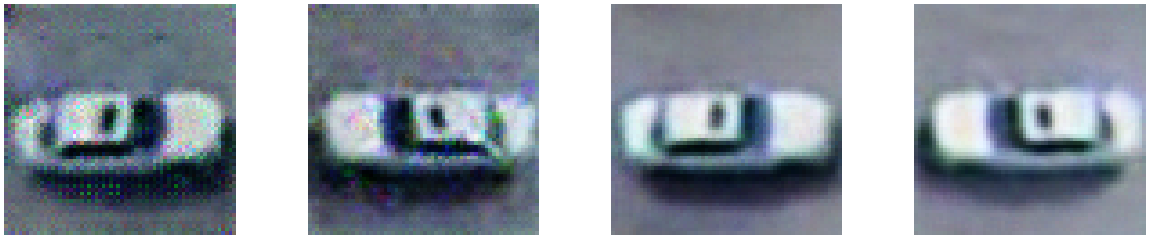
Figure 42: Examples of Synthetic Cars: Combined Dataset

All of the GANs were able to produce synthetic instances of the car. One interesting observation we made was some of the above cars originally captured by the moving platform which were thicker in diameter were produced thinner due to the GAN combining the slim shape of the car captured at the sideways view with the thick shape of the car captured from an overhead view. Those instances occured more when the GANs were conditioned on labels. Both of the WGAN variants showed no signs of random RGB pixels while RGB pixels were present in images generated by the two DCGAN implementations.

Table 12: Percent Change in Cars: Combined Dataset

GAN	Generated Instances	Percent Change
DCGAN	124	59.62%
cDCGAN	122	58.65%
WGAN	122	58.65%
cWGAN	86	41.35%

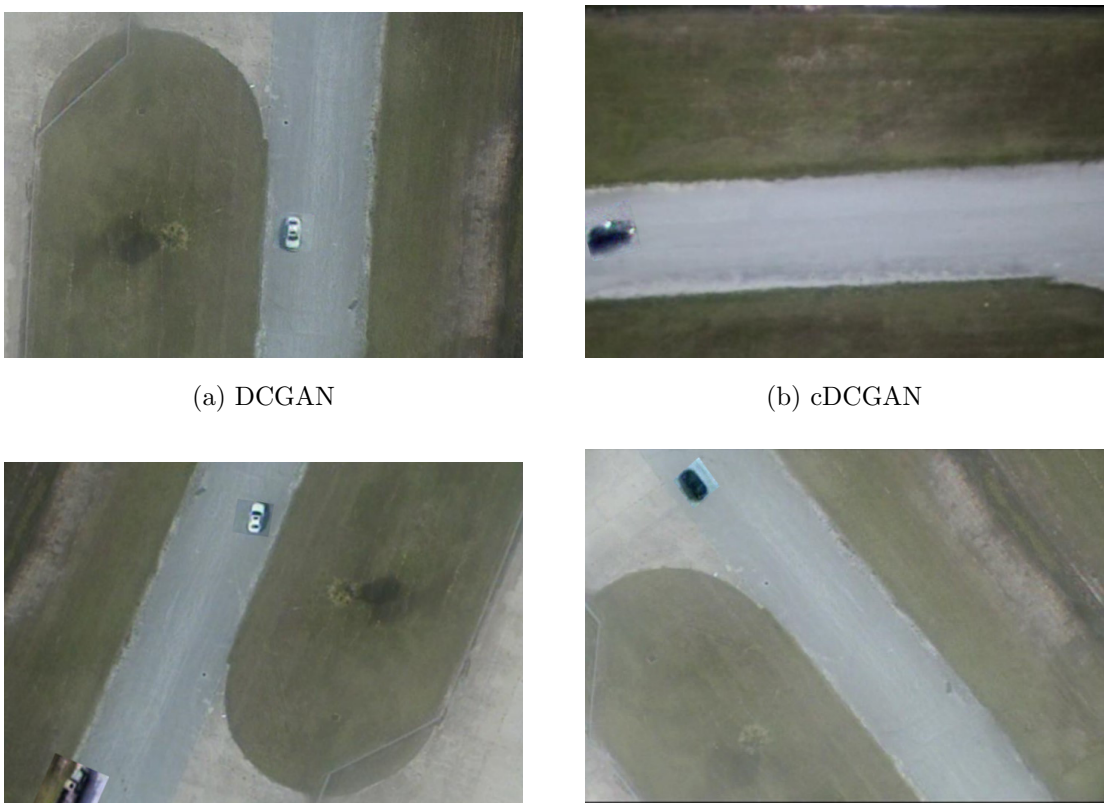
Consistent with the generation of the trucks, Table 12 outlines how all of the GAN variants produced approximately the same amount of new synthetic instances besides the cWGAN which was constrained by memory issues during its training process.



(a) DCGAN (b) cDCGAN (c) WGAN (d) cWGAN

Figure 43: Examples of Synthetic Cars: Combined Dataset

Figure 44 shows different examples of the generated images placed within the parent images for the combined dataset.



(a) DCGAN

(b) cDCGAN

(c) WGAN

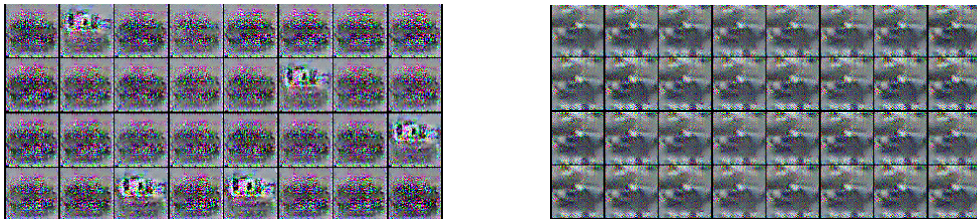
(d) cWGAN

Figure 44: Examples of Augmented Parent Images: Combined Dataset

4.5 Poor Quality Images

During the process of generating synthetic images, we deemed many samples poor quality and discarded them for use in augmenting the training sets. One speculation to why poor images may have been created is the generator was having success in tricking the discriminator (critic) at different times of the GAN training process with these poor samples and therefore continued to produce the inferior images. Another reason for the lower quality output of synthetic objects was the original images themselves. The frames provided were taken from a high altitude and had a lower resolution. The generator was learning to produce images from lower quality examples and due to the variance of the output of a GAN, some images were going to be much poorer in

quality than others.

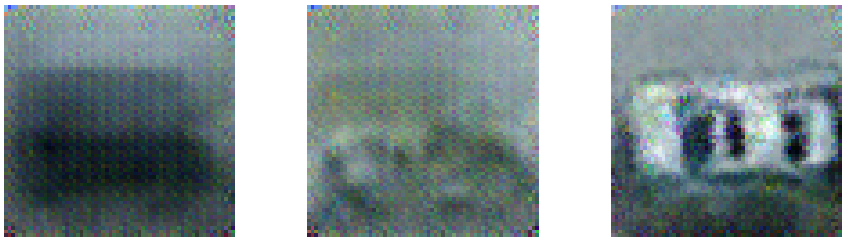


(a) Example 1 (b) Example 2

Figure 45: Examples of Mode Collapse for the DCGAN Augmentation from the Combined Dataset

One problem occurring during the training of the GANs was their instability in producing images resembling the original training set. The GAN architecture which had the most difficulty in providing the desired generated images was the DCGAN. The DCGAN had to be retrained various times to get the desired output. Figure 45 shows two examples of modal RGB noise produced by the DCGAN. This was a "collapse" in the GANs training process and possible explosion of the gradients in the discriminator and generator loss functions. In Figure 45a, the generator is showing to still make attempts at producing white cars and the black circles are understood to be possible reproductions of the van. In Figure 45b, the DCGAN was starting to learn how to generate the white car but could not create the right images to convince the discriminator into believing they were real, leading to a singular, poor output.

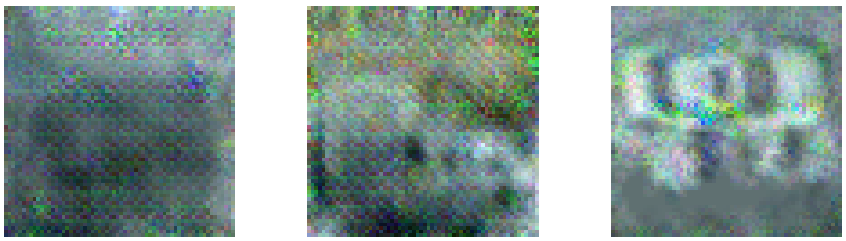
4.5.1 Dataset 1



(a) Van (b) Truck (c) Car

Figure 46: Examples of Poor Quality DCGAN Images: Dataset 1

Examination of the poor quality images from the DCGAN trained on Dataset 1 displays its failure in producing the truck and provides context for the difficulty the GAN had in producing the van and the car. There were no clear features for the truck and the generator began to use a smearing affect of the colors representing the truck in an effort to trick the discriminator. The van had many instances of black randomness and the instances of the car not accepted included cars with indistinguishable features at first glance.

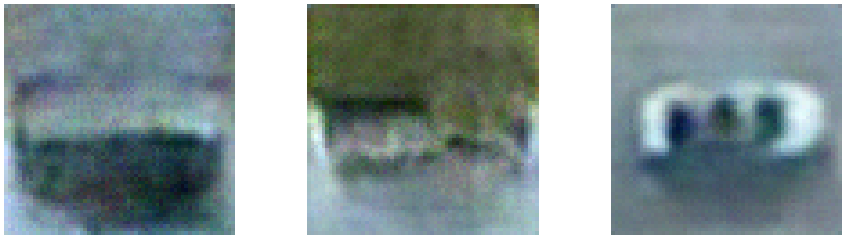


(a) Van (b) Truck (c) Car

Figure 47: Examples of Poor Quality cDCGAN Images: Dataset 1

The poor images for the cDCGAN closely aligned with the DCGAN output, however, this GAN was more capable of learning the car. The poor car instances showed a regularly shaped car but generally had problems of reproducing other "car" features, as shown by Figure 47c, which deemed those synthetic objects unusable for training.

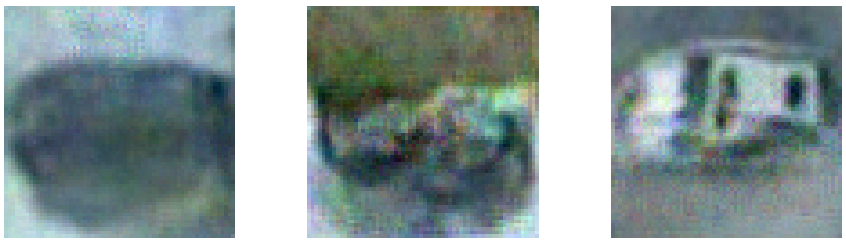
The same randomized technique of creating patches of colors similar to the original objects was used by the generator for creating the van and truck to try to deceit the discriminator if possible.



(a) Van (b) Truck (c) Car

Figure 48: Examples of Poor Quality WGAN Images: Dataset 1

The WGAN generated poor instances more visually close to acceptable use for dataset augmentation compared the DCGAN variants. The lower quality van and car instances began mimicking the original objects but still were not suitable to be placed in the frames during the augmentation phase. Figure 48b shows how the WGAN was beginning to learn to generate the truck with only a few original instances of the vehicle provided but most of the generated images were either smeared or were missing an element of the vehicle.



(a) Van (b) Truck (c) Car

Figure 49: Examples of Poor Quality cWGAN Images: Dataset 1

For the poor images of the cWGAN trained on Dataset 1, the van instances were closest to the higher quality images used for training. Due to their complexity,

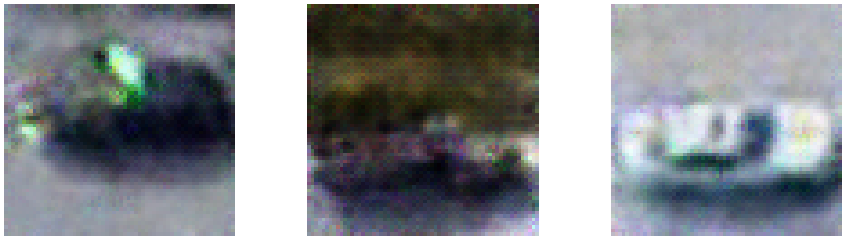
this variant was not able to fully capture how to generate the truck or car. While conditioning the WGAN added more stability in the training process, there was still a vast variance in the quality of images generated by the cWGAN.

4.5.2 Dataset 2

(a) Van (b) Truck (c) Car

Figure 50: Examples of Poor Quality DCGAN Images: Dataset 2

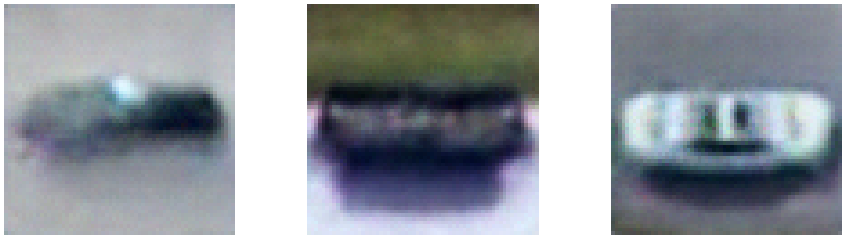
For the DCGAN trained on Dataset 2, one interesting type of synthetic image created was a truck with two fronts. The DCGAN added together two trucks facing different directions, creating the morphed synthetic instance in Figure 50b. For the van class, many of the images had the smearing pattern seen before which is attributed to the shorter learning time and lower quality images provided. The poor quality cars most closely resembled the original car but still showed too low of a resolution and indistinct features to be deemed acceptable for augmentation.



(a) Van (b) Truck (c) Car

Figure 51: Examples of Poor Quality cDCGAN Images: Dataset 2

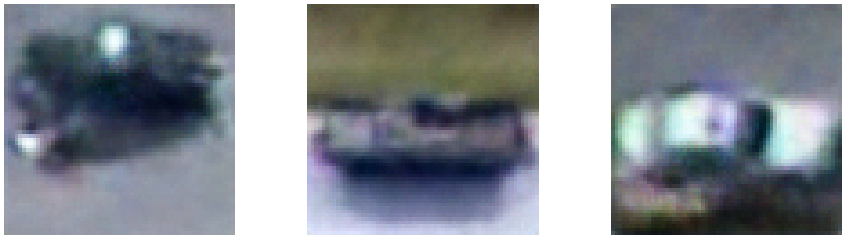
With the cDCGAN, the poor quality images all closely aligned with the object they were supposed to mimic. The van class had the necessary shape but still was covered with a blended finish of its features. The truck has the desired shape as well but is missing the vivid features required for acceptable use in the augmentation phase. The car class was the best generated class by the cDCGAN and the worst instances of the car were only considered poor due to their low resolutions. The shape and features of the car were acceptable for use during dataset augmentation.



(a) Van (b) Truck (c) Car

Figure 52: Examples of Poor Quality WGAN Images: Dataset 2

An example of the poor instances of the van showed how it was left unfinished by the generator of the WGAN. Another interesting morphed image of the truck was the inverse created by the DCGAN where the WGAN generator combined the two ends of a truck to generate a new object. Many of the poor car instances lacked the necessary detail or were unrealistic in architecture compared to the original cars in the training set.



(a) Van (b) Truck (c) Car

Figure 53: Examples of Poor Quality cWGAN Images: Dataset 2

The cWGAN showed to have some problems with the shading of the images, seen in the van and car instances. The cWGAN produced synthetic trucks but had problems of generating many unfinished samples. The truck in Figure 53b shows an example of missing features with the absence of the front cabin seen in the original trucks of the dataset.

4.5.3 Combined Dataset

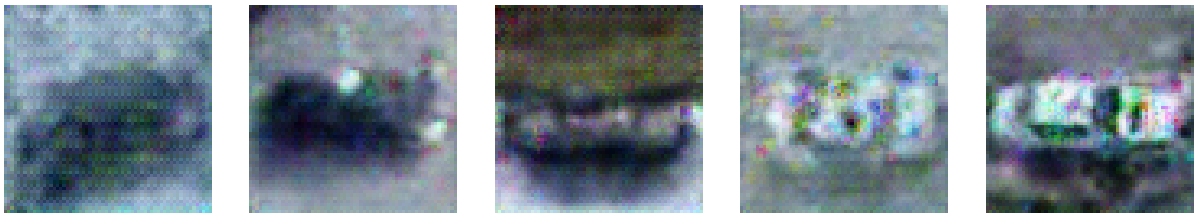
While the training sessions for each GAN had more instances for the generator to learn from, unsatisfactory images for the data augmentation phase were still produced. We concluded the more images provided to a GAN, the better quality of output of synthetic images.

(a) Van (1) (b) Van (2) (c) Truck (d) Car (1) (e) Car (2)

Figure 54: Examples of Poor Quality DCGAN Images: Combined Dataset

The first variation of the poor quality van generated by the DCGAN had resembling colors deriving from the truck and lacked continuity of features in its generation. The second style of van was generated with features originating in the white car which shows how the generator was attempting to create a combination of the two different vehicles. The poor quality truck instances generally had the correct shape but either were too low in resolution to be accepted for augmentation or were missing important features needed to be classified as a high quality synthetic truck. The main problems with the synthetic car objects were high levels of discoloration leading to the blending of the features of the generated cars. Some of the car objects began to adopt a grey

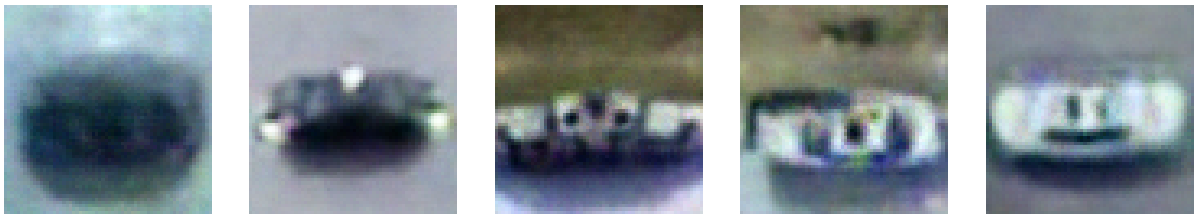
complexity as the generator was beginning to combine the colors of the car and van classes.



(a) Van (1) (b) Van (2) (c) Truck (d) Car (1) (e) Car (2)

Figure 55: Examples of Poor Quality cDCGAN Images: Combined Dataset

The main quality issue for the cDCGAN was blurriness in the generated images. Many of the instances for the first type of van lacked the detail needed to be considered acceptable for augmenting the original frames. For the second van style, most of the problems were faded details and poor resolution quality. When producing new trucks, if the generator forgot to add a part of the vehicle then the image was considered unusable for the augmentation phase. For the first type of car depicting an overhead view, the generator created vehicles too wide and blurry. For the second style of the car, some of the vehicles turned grey which was acceptable but if the picture itself was hazy then we did not place that car instance into the parent frames.

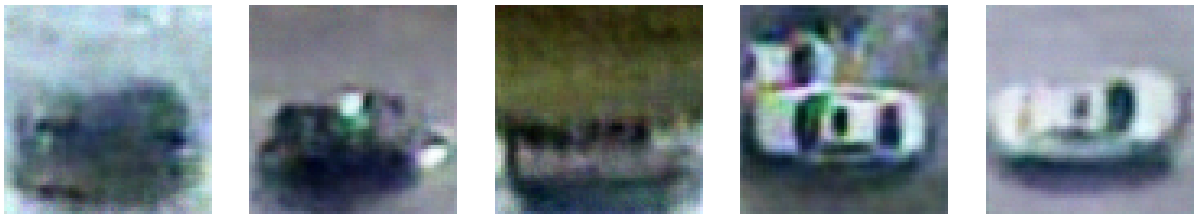


(a) Van (1) (b) Van (2) (c) Truck (d) Car (1) (e) Car (2)

Figure 56: Examples of Poor Quality WGAN Images: Combined Dataset

The poor quality images obtained from the WGAN had less issues of poor resolution quality and more issues of insufficient structural components of each generated

vehicle. For the van instance, the problems for the first type of van were its representation as a generic black circle. For the second style of van, some of the vans possessed the combination of the two fronts of the van, shown by Figure 56b. The truck objects had the same outcome where the generator of the WGAN produced trucks with two fronts. For the cars, most of the issues centered around their production involved the cars being either too wide or having a bent frame. The bending may be coming from the way the generator is combining the different angles from which the objects were captured by the moving platform in the original frames.



(a) Van (1) (b) Van (2) (c) Truck (d) Car (1) (e) Car (2)

Figure 57: Examples of Poor Quality cWGAN Images: Combined Dataset

The poor quality images of the cWGAN had problems commensurate to the WGAN. Most of the problems were structural and did not stem from issues in the resolution. One interesting creation the cWGAN began producing was placing two cars into an image which the GAN was never given examples of to use for training. This shows the power of the GAN to learn how to incorporate more than one car into each of the images. While unsuccessful in its attempts, future research can include investigating if the number of objects in an image which a GAN is producing can be mapped to a vector in the latent space.

4.6 Generative Adversarial Network (GAN) Training Takeaways

Brightness was one of the main reoccurring problems when recreating darker objects. The shadow of an object occurring from its position to the sun would sometimes force the generator to create an inaccurate representation of the object since it blended the shadow into the darker figures, creating unrealistic objects.

Another assertion we made from our observations for GAN training is it is much harder to train a GAN with a limited number of images. Even with an adequate amount of images for training a model, the research showed how a GAN will have difficulty learning a certain class in the dataset if the class is underrepresented. This can be seen in the training of a GAN using Dataset 1 with the truck instances. The truck class provided a limited number of training instances and none of the GAN variants generated the required high quality standards of the original truck images compared to the car and van instances.

Due to the lower resolution images gathered, the synthetic images from the GANs also had a lower quality resolution in nature. While some of the images looked to be in poor condition, comparing them to the real images they were supposed to represent showed they were very similar in detail. The GANs utilizing WGAN architecture produced finer quality images than those adapted to the DCGAN architecture. We inferred the added stability from the Wasserstein Loss Function and changes to the training of the critic led to the generation of sharper contrasts of features and colors in the synthetic images. There were multiple instances of the DCGAN producing images containing random RGB pixel spots which can be attributed to the decrease in stability of the DCGAN architecture compared to the WGAN architecture. Further research efforts should be directed towards increasing GAN stability and synthetic image quality assessment.

V. Results and Discussion

Preamble

This chapter provides the results of the You Only Look Once Version 4 - Tiny (YOLOv4-Tiny) Object Detection Model training. We trained the model five different ways: first with the original datasets only, and second with the original datasets augmented by each of the four Generative Adversarial Networks (GANs) individually. This resulted in sixty training trials, all of which were compared against the original test set.

5.1 Model Training: Dataset 1

We trained a model on Dataset 1 followed by training models on Dataset 1 augmented with each of the four GANs individually. The Conditional Wasserstein Generative Adversarial Network (cWGAN) augmented dataset produced the best performing model with a Mean Average Precision (mAP) of 94.78% and an average Intersection over Union (IoU) value of 67.24%. Compared to the model trained on Dataset 1 with no augmentation, these values were greater by 6.59% and 2.90%, respectively.

5.1.1 Dataset 1: No Augmentation

As noted in Chapter III, we trained multiple models using learning rate tuning. Table 13 shows the highest performing trained YOLOv4-Tiny Model using Dataset 1 with no data augmentation. The most efficient model was trained with a learning rate of 0.001 with 6,000 iterations, producing a mAP of 88.19%, a Complete Intersection over Union (CIoU) loss of 0.0400, an average IoU of 64.34%, a precision rate of 0.81, a

recall rate of of 0.82, a F1-score of 0.81, and True Positive (TP), False Positive (FP), and False Negative (FN) detections of 140, 33, and 30, respectively.

Table 13: YOLOv4-Tiny Model Training Results: Dataset 1 Best Model

Learning Rate	**mAP**	**CIoU Loss**	**Avg.IoU**	**Precision**	**Recall**	**F1-Score**	**TP**	**FP**	**FN**
0.001	88.19%	0.0400	64.34%	0.81	0.82	0.82	140	33	30

For each individual class, the model had the most difficulty with localizing and classifying the truck objects within each frame. It exhibited similar performance for the car and van instances within the images.

Table 14: YOLOv4-Tiny Model Training Results by Class: Dataset 1 Best Model

Class	**mAP**	**TP**	**FP**
Van	99.67%	48	9
Truck	65.26%	45	22
Car	99.60%	47	2

A chart mapping mAP and CIoU Loss was collected for each training iteration to visually inspect how the model trained overtime. The model did not show much improvement over the course of its training, suggesting the need for augmentation and hyperparameter tuning to achieve performance improvement.

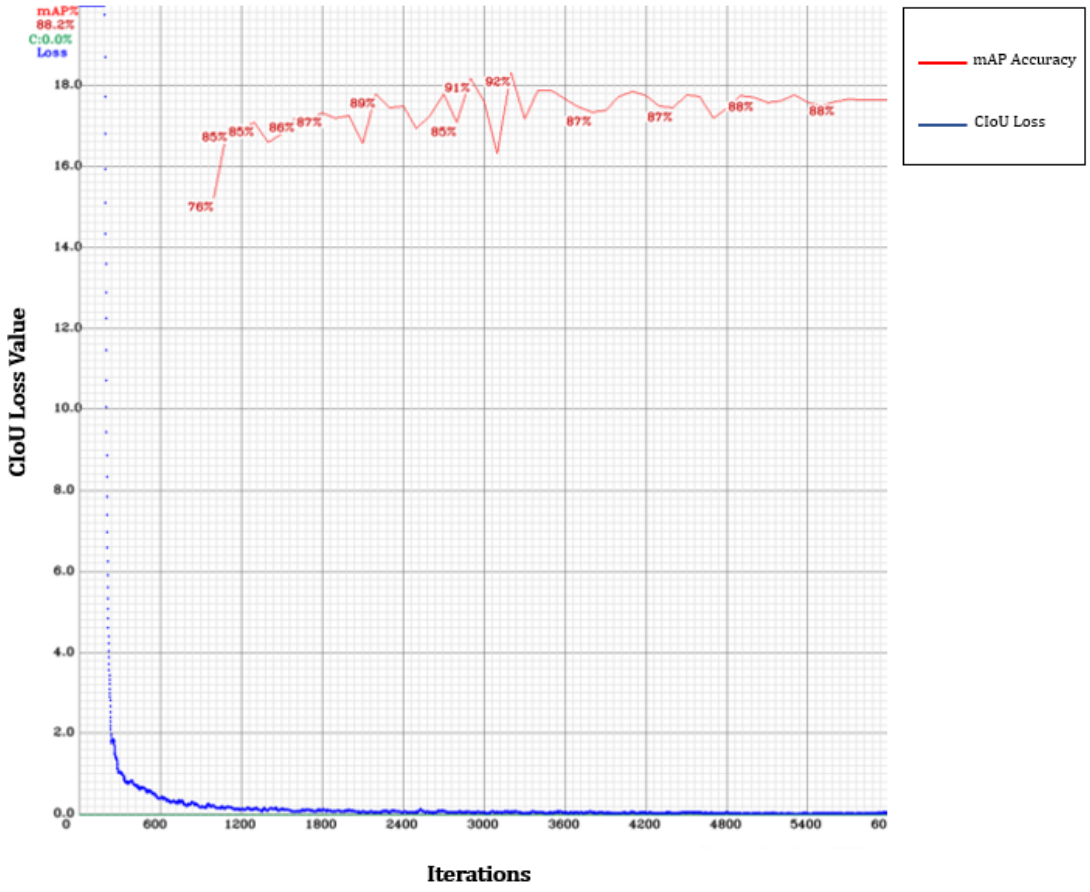


Figure 58: mAP vs. CIoU Loss: Dataset 1

5.1.2 Dataset 1: GAN Augmentation

Table 15: YOLOv4-Tiny Model Training Results: Dataset 1 + GAN Images Best Models

GAN	Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
DCGAN	0.0001	90.55%	0.1044	65.36%	0.86	0.81	0.83	138	23	32
cDCGAN	0.001	94.25%	0.1219	65.49%	0.86	0.91	0.89	155	25	15
WGAN	0.001	90.95%	0.0518	66.13%	0.81	0.81	0.81	138	32	32
cWGAN	**0.0001**	**94.78%**	**0.1223**	**67.24%**	**0.86**	**0.84**	**0.85**	**145**	**24**	**25**

The best performing model from each Generative Adversarial Network (GAN) augmented dataset is outlined in Table 15. The GAN producing the most effective model was the cWGAN. Using the cWGAN augmented images led to a 6.59% increase

in mAP compared to the baseline model and the average IoU accuracy also increased by 2.90%. Precision, recall, and the F1-Score increased by 4.00%, 2.00%, and 3.00% respectively. The number of TP detections increased by 5 while the number of FP detections decreased by 9, and the number of FN detections decreased by 5.

Table 16: YOLOv4-Tiny Model Training Results by Class: Dataset 1 +cWGAN Images

Class	**mAP**	**TP**	**FP**
Van	95.41%	46	7
Truck	88.93%	55	17
Car	100.00%	44	0

When the cWGAN augmented training set was used, the mAP of the truck increased by 23.67%. TP detections increased by 10 while FP detections decreased by 5. GAN augmentation produced a 0.40% increase in the mAP for the car class with a decrease of 3 TP detections. The decline in TP detections was offset by 0 FP detections. For the van class, training with the GAN augmented dataset had an adverse effect on the model's performance, dropping mAP by 4.26%.

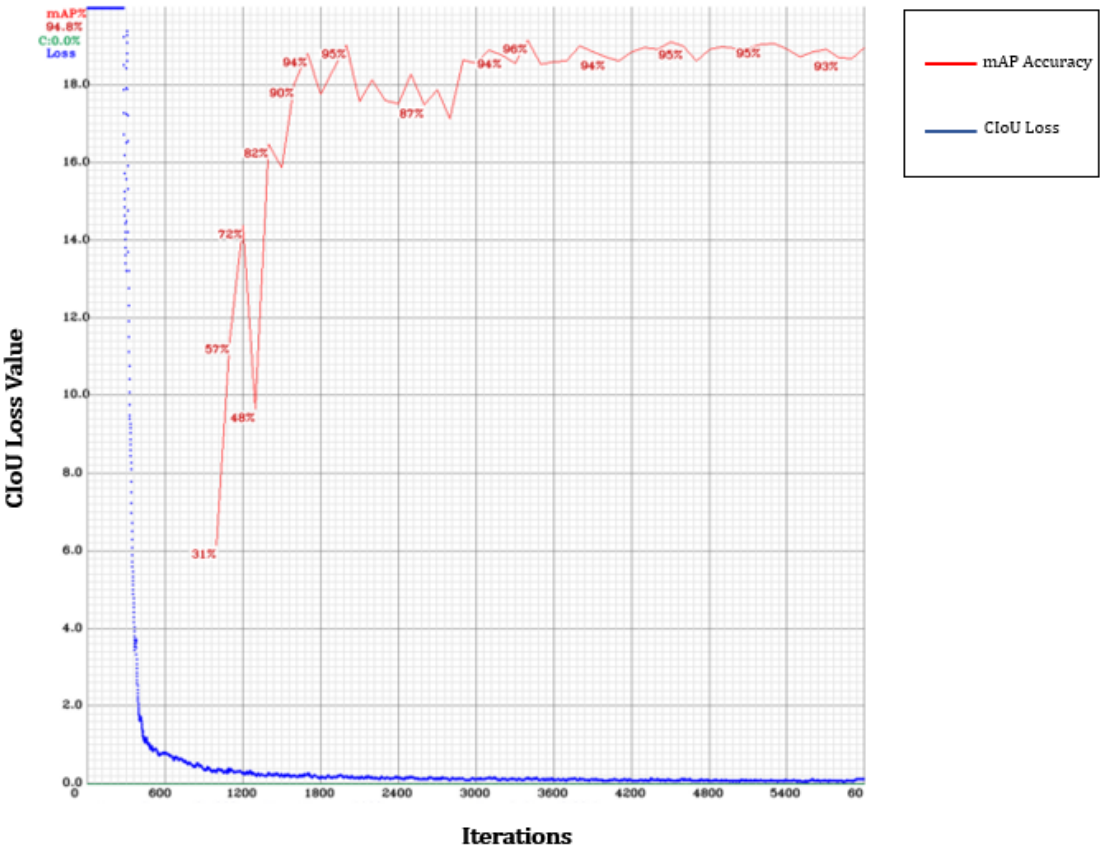


Figure 59: mAP vs. CIoU Loss: Dataset 1 + cWGAN

The mAP and CIoU loss curves show consistent training over time for the model with a few spikes in mAP. The model begins to plateau in training at approximately 5,400 iterations, suggesting the training of the network had reached convergence.

5.2 Model Training: Dataset 2

We trained a model on Dataset 2 followed by training models on Dataset 2 augmented with each of the four GANs individually. The Deep Convolutional Generative Adversarial Network (DCGAN) augmented dataset produced the best performing model with a mAP of 89.13% and an average IoU score of 71.89%. Compared to the model trained on Dataset 2 with no augmentation, these values were greater by

15.76% and 9.60%, respectively.

5.2.1 Dataset 2: No Augmentation

The best performing model used a learning rate of 0.0001. The model yielded a mAP of 73.37%, a CIoU loss of 0.1960, an average IoU rate of 62.29%, a precision rate of 0.79, a recall rate of 0.56, a F1-Score of 0.66, and TP, FP, and FN detections of 96, 25, and 74, respectively.

Table 17: YOLOv4-Tiny Model Training Results: Dataset 2 Best Model

Learning Rate	**mAP**	**CIoU Loss**	**Avg. IoU**	**Precision**	**Recall**	**F1-Score**	**TP**	**FP**	**FN**
0.0001	73.37%	0.1960	62.29%	0.79	0.56	0.66	96	25	74

The best performing model did not do well on the van instances, obtaining a mAP value 56.70% lower than Dataset 1. The cause of this negative effect may stem from how Dataset 1 contained more instances of the van which were similar to the vans in the test set. Compared to Dataset 1, the model did a much better job of learning and finding the truck instances within the parent images, achieving a mAP 24.95% higher than Dataset 1.

Table 18: YOLOv4-Tiny Model Training Results by Class: Dataset 2 Best Model

Class	**mAP**	**TP**	**FP**
Van	42.97%	15	0
Truck	90.21%	41	1
Car	86.93%	40	24

The chart of the mAP and CIoU loss shows the model's mAP beginning to plateau at approximately 3,000 iterations. CIoU loss continuously decreased with a slight increase towards the end of training.

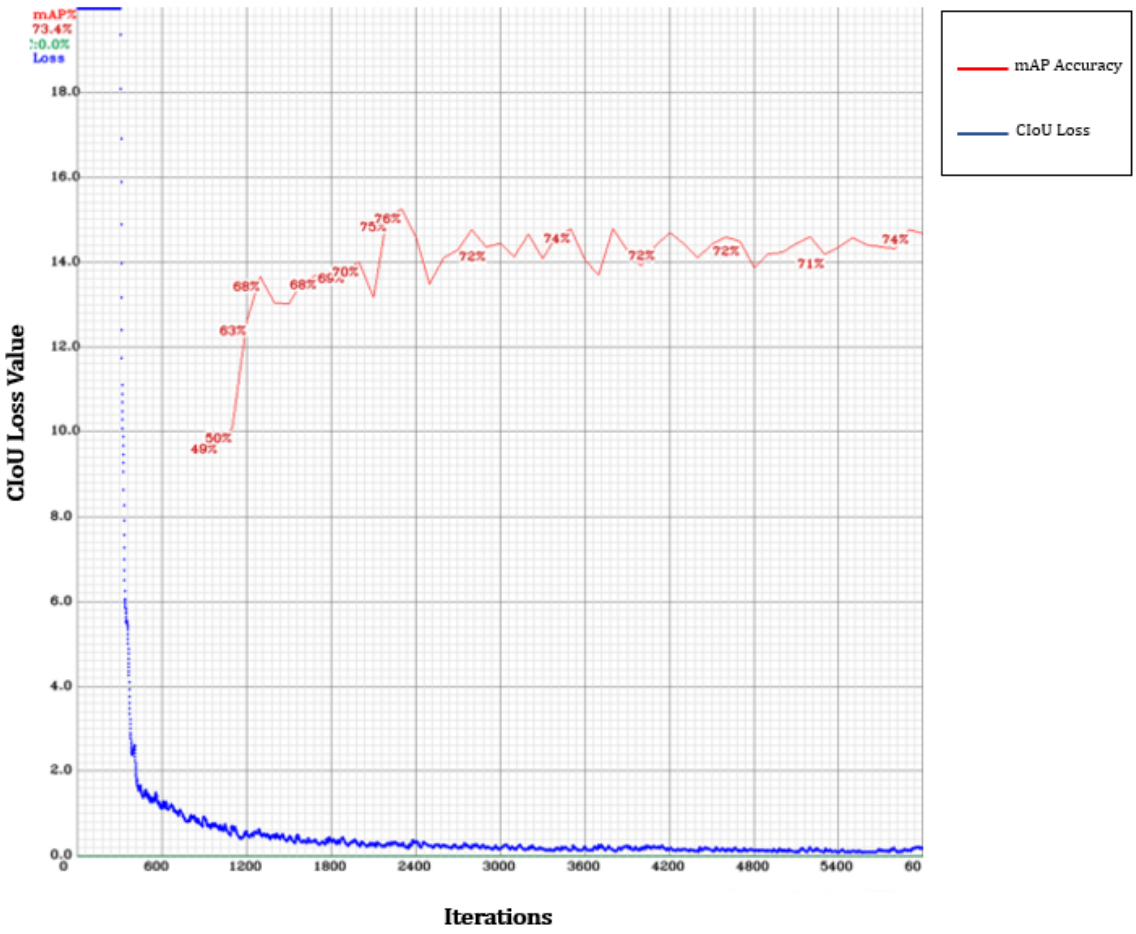


Figure 60: mAP vs. CIoU Loss: Dataset 2

5.2.2 Dataset 2: GAN Augmentation

The augmented training set leading to the highest increase in overall performance was the collection of images augmented with the DCGAN and trained with a learning rate of 0.0001. The only downfall to this model was the higher value of CIoU loss which we accredited to the use of more diverse and different images compared to the test set.

Table 19: YOLOv4-Tiny Model Training Results: Dataset 2 + GAN Images Best Models

GAN	Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
DCGAN	**0.0001**	**89.13%**	**0.2744**	**71.89%**	**0.95**	**0.72**	**0.82**	**122**	**7**	**48**
cDCGAN	0.0001	74.17%	0.2744	61.92%	0.81	0.58	0.68	100	24	70
WGAN	0.0001	73.40%	0.2675	64.09%	0.82	0.58	0.68	99	21	71
cWGAN	0.0001	73.04%	0.2463	53.50%	0.68	0.54	0.60	92	44	78

The mAP increased by 15.76% and the average IoU rate increased by 9.60%. Precision, recall, and the F1-Score all increased by 16.00%. The number of TP detections increased by 26, while FP and FN detections decreased by 18 and 26, respectively. Although the mAP for the truck decreased by 1.45%, higher mAP values were achieved for the van and car classes. The mAP for the van class increased by 38.24%, with TP detections increasing by 15. FP detections did increase for the van class from 0 to 3. For the car class, mAP increased by 10.48%, TP detections increased by 6, and FP detections decreased by 20.

Table 20: YOLOv4-Tiny Model Training Results by Class: Dataset 2 + DCGAN Images

Class	mAP	TP	FP
Van	81.21%	30	3
Truck	88.76%	46	0
Car	97.41%	46	4

The graph of the mAP and CIoU loss curves shows an increasing rate over time, however, there are instances of dramatic drops in mAP. The use of augmented images added some instability during the midpoint iterations in the training of the model. While there was more variance in model training, the mAP begins to plateau towards the end of the cycle showing a stable final model.

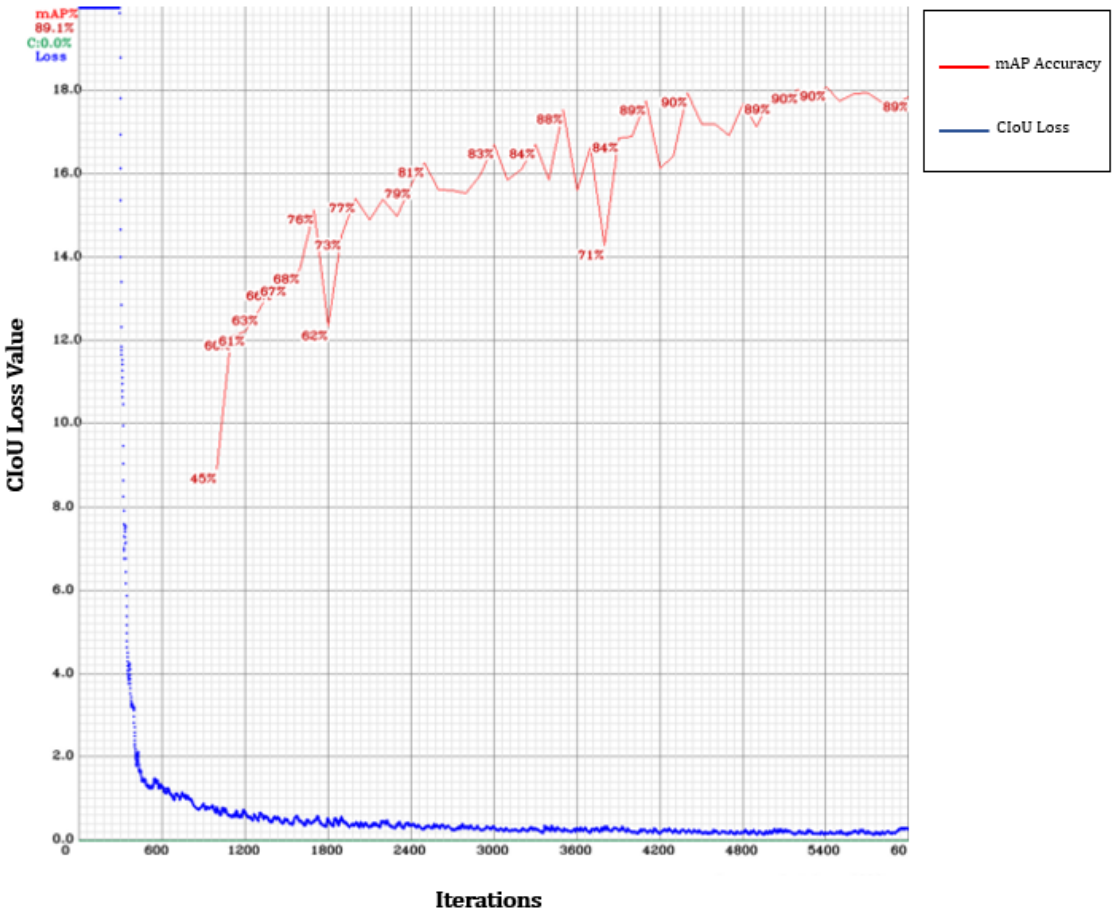


Figure 61: mAP vs. CIoU Loss: Dataset 2 + DCGAN

5.3 Model Training: Combined Dataset

We trained a model on the combined dataset followed by training models on the combined dataset augmented with each of the four GANs individually. For the combined dataset, the cWGAN augmented dataset produced the best performing model with a mAP of 100.00% and an average IoU score of 80.39%. Compared to the model trained on the combined dataset with no augmentation, these values were greater by 1.47% and 3.81%, respectively.

5.3.1 Original Dataset: No Augmentation

The best performing model using the training set with no augmentation had a learning rate of 0.001. The model concluded training with a mAP of 98.53%, an average IoU of 76.58%, a precision rate of 0.94, a recall rate of 0.98, a F1-Score 0.96, and TP, FP, and FN detections of 167, 11, and 3, respectively.

Table 21: YOLOv4-Tiny Model Training Results: Combined Dataset Best Model

Learning Rate	**mAP**	**CIoU Loss**	**Avg. IoU**	**Precision**	**Recall**	**F1-Score**	**TP**	**FP**	**FN**
0.001	98.53%	0.0793	76.58%	0.94	0.98	0.96	167	11	3

The mAP values for each of the objects of interest were all in the upper 90% range, showing the strengths of increasing the size of a dataset for increasing a model's effectiveness. Table 22 outlines the efficiency of the model in its localization and classification of each of the objects. The model was strongest in its ability to detect trucks with a 99.89% mAP value and 1 FP detection for every 25 TP positive detections.

Table 22: YOLOv4-Tiny Model Training Results by Class: Combined Dataset Best Model

Class	**mAP**	**TP**	**FP**
Van	97.47%	45	2
Truck	99.89%	75	3
Car	98.24%	47	6

The graph of the mAP and CIoU loss curves shows the model stabilizing at around 2,400 iterations. Steady decreases in CIoU loss also show how the model was properly training and not overfitting or underfitting.

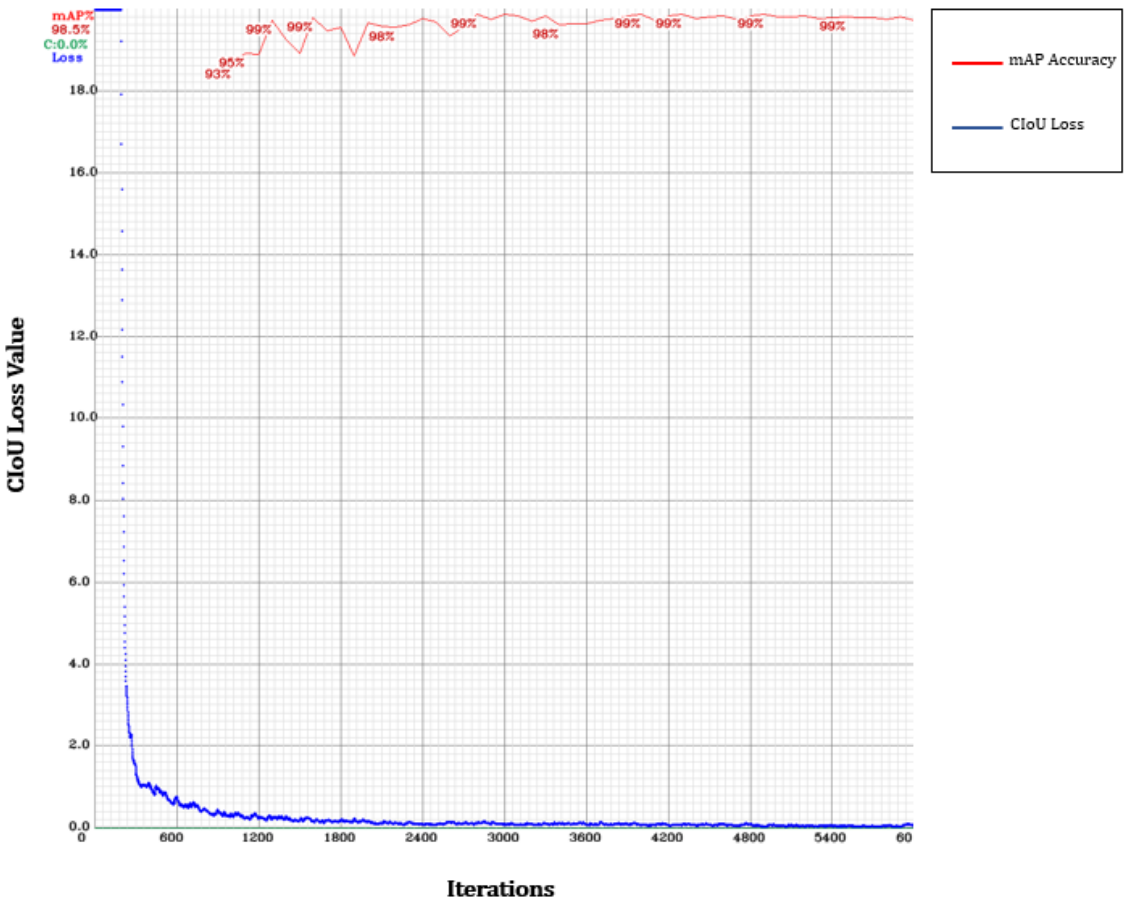


Figure 62: mAP vs. CIoU Loss: Combined Dataset

5.3.2 Combined Dataset: GAN Augmentation

When trained on datasets augmented with images from GANs, most of the models achieved superior results with high mAP values. The CIoU loss was therefore taken more into consideration when selecting the best model from each trial.

Table 23: YOLOv4-Tiny Model Training Results: Combined Dataset + GAN Images Best Models

GAN	Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
DCGAN	0.00261	100.00%	0.0765	80.08%	0.96	1.00	0.98	170	0	7
cDCGAN	0.00261	99.26%	0.0801	79.76%	0.95	0.99	0.97	169	9	1
WGAN	0.00261	100.00%	0.0678	81.56%	0.97	1.00	0.99	170	5	0
cWGAN	**0.00261**	**100.00%**	**0.0595**	**80.39%**	**0.96**	**1.00**	**0.99**	**170**	**8**	**0**

The selected best model achieved when using training sets augmented with images from GANs was the training set supported with synthetic images generated by the cWGAN. Comparing the results to the original model, mAP increased by 1.47% while CIoU loss decreased by 1.98%. The average IoU also positively benefited from the augmentation with an increase of 3.81%. Precision increased by 2.00%, recall increased by 2.00% and the F1-Score increased by a total of 3.00%. In addition, the number of TP detections increased by 3 and FP detections were 0 with a decrease of 11 false detections. The only metric negatively impacted from the addition of images generated by a GAN was the FN metric, which increased by a value of 4 to 7 detections. This negative impact was offset by the increase in TP detections.

All three of the object classes had individual mAP values of 100.00%. For the van class, mAP increased by 2.53% while TP detections were a perfect 48. FP detections were reduced by 2 to a value of 0. The truck class had a mAP reduction of 0.11% with no increase in the TP detections made on trucks and an additional two FP detections. The mAP for the car class increased by 1.76%. The car class also had no increase in TP detection but had a 2 unit decline in FP detections.

Table 24: YOLOv4-Tiny Model Training Results by Class: Combined Dataset + cWGAN Images

Class	**mAP**	**TP**	**FP**
Van	100.00%	48	0
Truck	100.00%	75	5
Car	100.00%	47	3

While the mAP and CIoU loss graph shows a plateau in mAP at 100.00%, GAN augmentation can be seen to illicit some instability in the training progress. Instability can be observed at the huge decline in mAP at epoch 1,200. More extensive hyperparameter tuning may be necessary to stabilize the models trained with GAN augmented aerial image training sets.

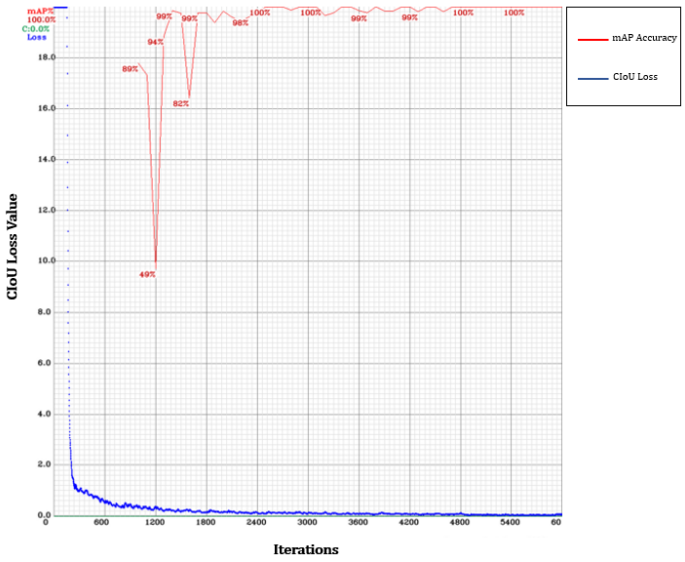


Figure 63: mAP vs. CIoU Loss: Combined + cWGAN

5.4 Discussion

The augmentation of the parent images with synthetic child objects resulted with mAP collectively increasing by 6.75% and average IoU increasing by 4.13%, on average. The benefits for Dataset 1, on average, were mAP increases of 8.80% and average IoU increases of 4.67%. The performance of the model trained on Dataset 2 received the biggest positive impact. mAP increased by 10.69% on average with the introduction of GAN augmentation and the average IoU across all trained models increased by 5.17% with augmentation. For the size of datasets, GAN augmentation had a much bigger impact on smaller datasets. This was due to the imbalance problems those datasets have relative to not enough samples and diversity for model training.

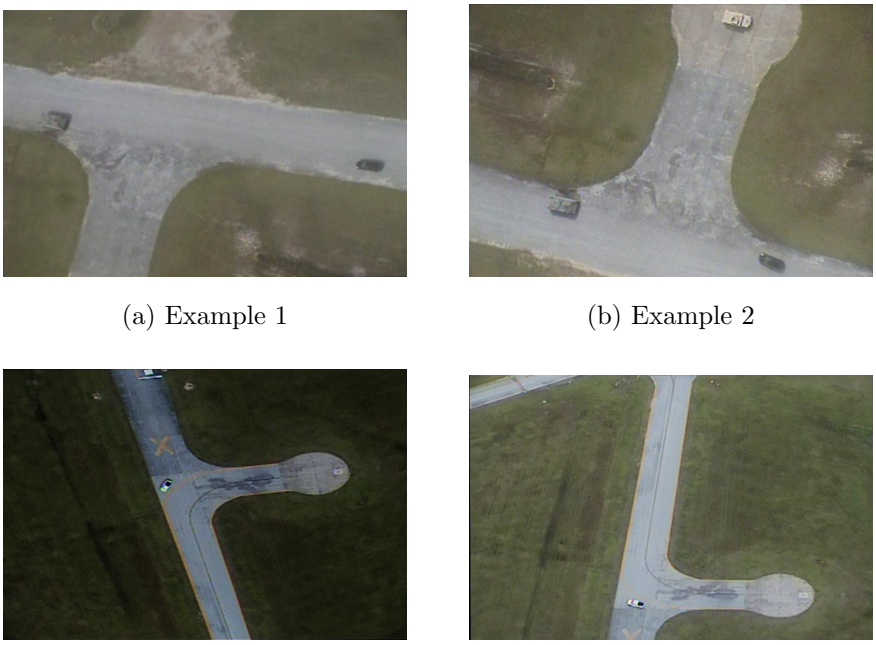
The combined dataset was least effected by GAN augmentation. mAP collectively increased by 0.75% on average and average IoU increased by 2.55% on average. The combined dataset received increases in its accuracy metrics, but also saw more FP detections for the trucks when the addition of augmented images were used for training.

VI. Follow-on Experiments

Preamble

After concluding the sixty training trials of the You Only Look Once Version 4 - Tiny (YOLOv4-Tiny) Object Detection Model with the original dataset, initial results prompted three more experiments. First, we created a new "Alien Test Set" (i.e. a test set unrelated to either of the training sets) for use in training to increase the diversity of the testing set. Second, we applied aggressive hyperparameter tuning to counter perceived over training witnessed during Experiment I. Finally, we used a blended multi-Generative Adversarial Network (GAN) approach to augment the datasets.

6.1 Experiment I: Alien Test Set



(a) Example 1

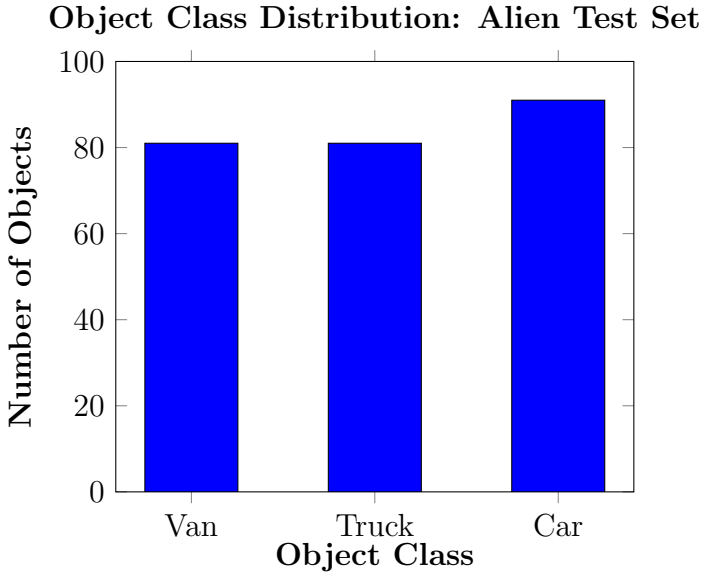
(b) Example 2

(c) Example 3

(d) Example 4

Figure 64: Examples of Alien Test Set Images

We created an Alien Test Set with images of vehicles traveling along a different road completely unrelated to the original training and test sets. We selected images containing instances of objects labeled as "van", "truck", and "car" to investigate the robustness of the trained detection model on new and different instances of objects from those classes. The samples in Figure 64 show the different types of frames included within the Alien Test Set.



There was a total of 81 vans, 81 trucks, and 91 cars in the new test set. The most dramatic differences in the features of the objects in the training set and Alien Test Set were between the truck and car classes.

(a) Van (b) Truck (1) (c) Truck (2) (d) Car

Figure 65: Examples of Objects in the Alien Test Set

The benchmark learning rate used for Experiment I was 0.001. We determined

this value after examining the results of the training trials for the YOLOv4-Tiny Model and its object detection capabilities on the Original Test Set. All of the training sets containing real and synthetic images from the Combined Parent Dataset were used for Experiment I. After conclusion of the experiment, the performances of the models using the augmented datasets were compared to the performance of the model with the original dataset. Any GAN augmented training set leading to the model outperforming the model using the original combined training set was deemed a "success" and further experiments using training sets augmented from the supporting GAN variant were halted.

6.2 Experiment II: Alien Test Set- Varying Hyperparameters

After the termination of Experiment I, we designed a new experiment for the inferior performing models using certain Generative Adversarial Networks (GANs) augmented datasets which varied the hyperparemters of the YOLOv4-Tiny Object Detection Model to see if there were other factors influencing performance. Experiment II was designed around the hypothesis: if a GAN augmented training set did not enhance a model's performance on the Alien Test Set, then the augmented training set was causing the model to overfit on the original dataset. We determined the model was overfitting on learning the objects deriving from the original dataset. This pattern was observed in the datasets augmented with images from the Conditional Deep Convolutional Generative Adversarial Network (cDCGAN), Wasserstein Generative Adversarial Network (WGAN), and Conditional Wasserstein Generative Adversarial Network (cWGAN). Only the datasets augmented with instances from these GANs were used for the remaining experiments. The hyperparameters altered were the learning rate and number of iterations. The baseline learning rate for Experiment II was 0.00261. The first collection of trials consisted of only changing the

learning rate. The learning rate was either increased by 50% or decreased by 50% when its value was altered.

The second assemblage of trials used a constant learning rate of 0.00261 but had changes in the number of epochs hyperparameter to measure the effect of varying iterations for object detection model training. 2,000 and 4,000 epochs were used to investigate if better performance could be achieved from the original 6,000 iterations. The hypothesis we constructed revolved around the notion of decreasing the number of epochs gives the model less time to learn to the features of the original dataset and forces the model to be more robust to never before seen instances.

The final collection of trials consisted of the combination of differing learning rates and number of epochs across each training sequence. The third variation of Experiment II had the goal of finding an optimal mix of the two hyperparameters which produced the best model trained on an augmented training set. The learning rate 0.00261 was decreased by 50% to 0.001305 as well as increased by 50% to .005220. For each of the two learning rates, the number of epochs was set to 2,000 and 4,000 iterations.

6.3 Experiment III: Blended Augmented Training Set

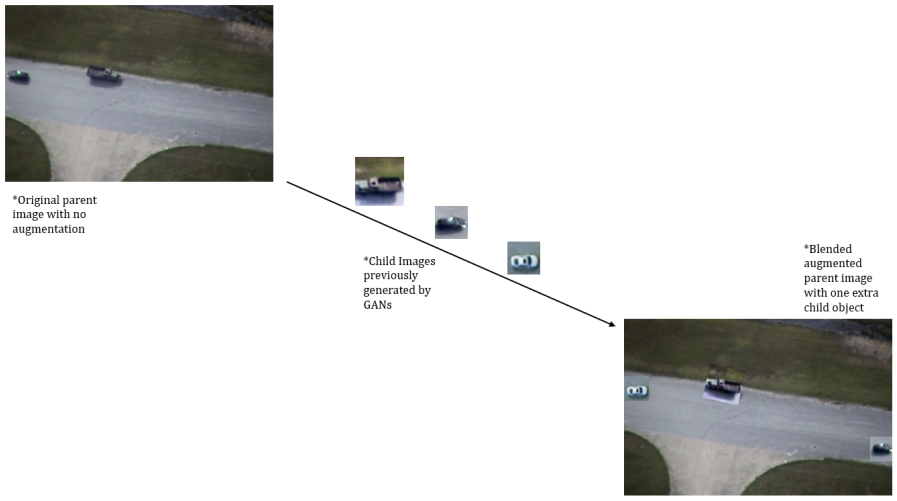


Figure 66: Blended Augmentation: Phase 1 Process

The last augmentation attempt for diversifying the original training set used images from all four of the GANs concomitantly for creating a new augmented training set. For the first phase of the blended augmentation, we doubled the dataset and the original process of covering the images was executed. An additional component for replacing all of the native objects was the addition of one extra entity to each parent image. For covering the original images, we made efforts to have an object from a different class cover an original figure. One limitation to this technique was how the van and car could not always cover the truck, therefore, we had augmented frames where we placed the truck in the same position in the parent image which the vehicle originally presided. Due to constraints of the amount of images and to reduce the risk of greater dataset imbalance, some images received more than one extra object, as well as synthetic vans and cars covering their real world child images.

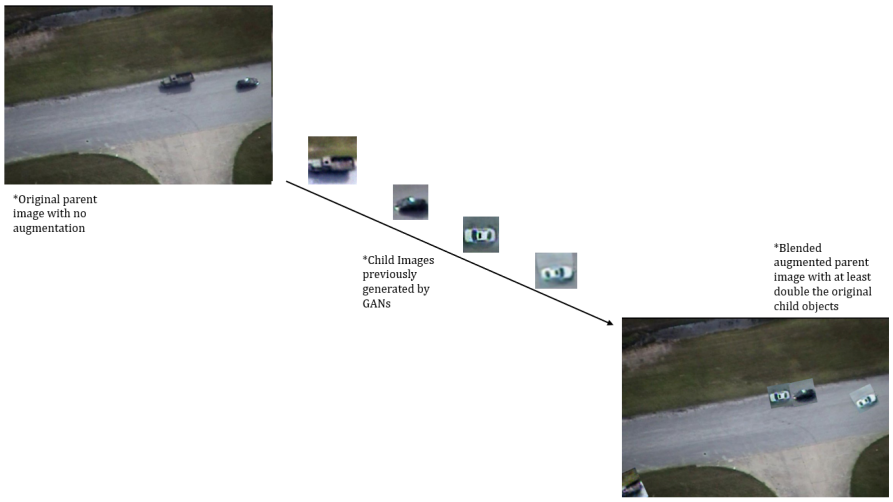


Figure 67: Blended Augmentation: Phase 2

After the dataset was doubled, phase two of the augmentation process sought to at least double the number of objects within each parent image. Due to unequal amounts of saved synthetic images, some of the new frame instances had more than a 100% increase in the number of the objects contained within their boundaries. We used the same previous technique to move objects from different parts of the image where they originally were captured by the Unmanned Aerial Vehicle (UAV), however, this was not feasible in some instances to ensure the original objects were covered.

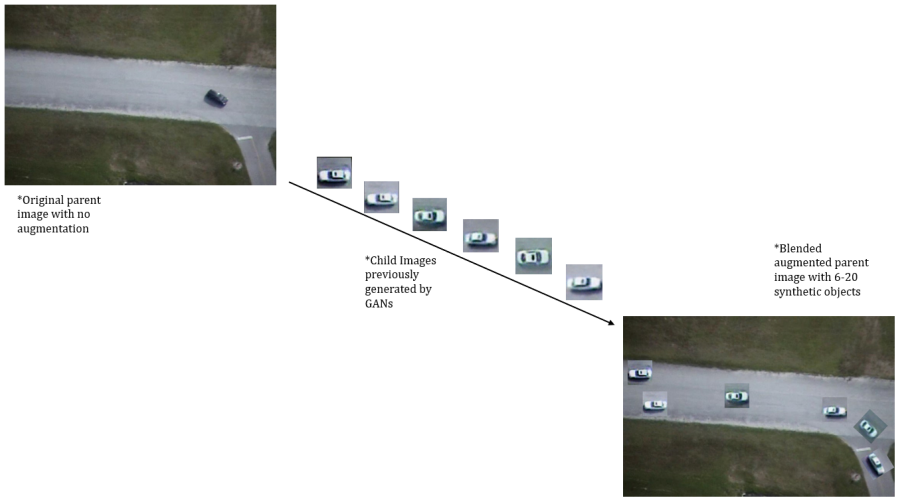
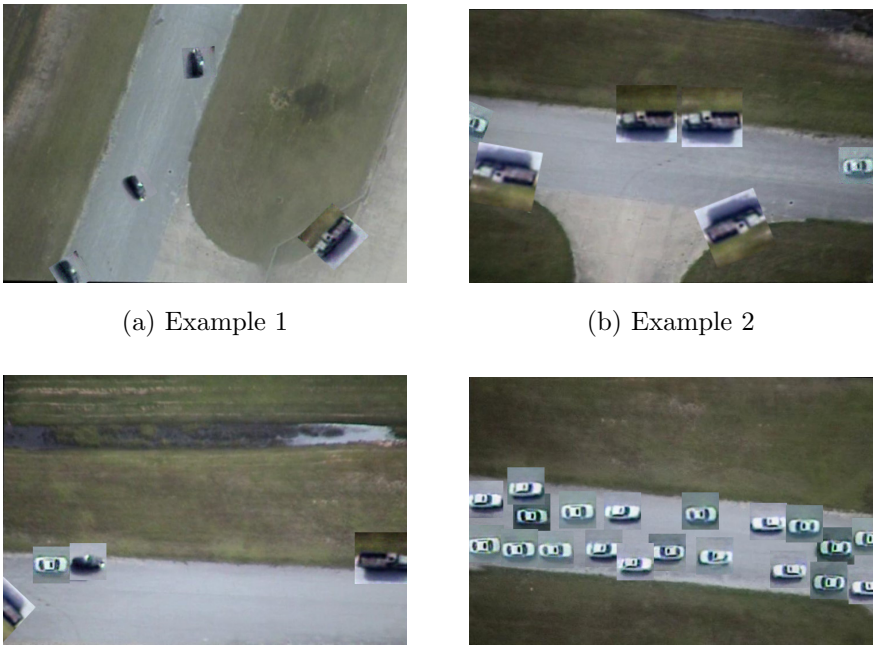


Figure 68: Blended Augmentation: Phase 3

The van class had the least number of synthetic replications, therefore, phase three of the blended augmentation process included a minority of batches containing anywhere from six to twenty cars and six to twenty trucks. These expansive techniques added a wide range of diversity across the new training set. Once the blended augmentation was complete, we trained the YOLOv4-Tiny Model with the learning rates of 0.00261, 0.001, 0.0001, and 0.00001 for 6,000 iterations on the original and Alien Test Sets.

6.3.1 Blended Augmentation

As mentioned, the last augmentation variation was combining the child objects from all four of the GANs to create one, blended training set. This augmentation attempt led to the biggest increase in number of images and object instances for all of the training set transformations. The total number of objects in the blended augmented dataset was 2,506. The van class had the largest percent change increase of 398.45% with the addition of 579 new instances to the parent frames. The car class had the second largest addition of synthetic images with an increase of 578 new objects or a 387.50% change increase. The truck class had the smallest percent change of 350.00% in the parent images. Even with the lowest percent change in number of objects in the training set, trucks were now the highest number of objects spread throughout all of the frames with the addition of 660 synthetic instances and a total of 924 instances. The blended augmented dataset was comprised of 30.88% vans, 36.92% trucks, and 32.20% cars.



(a) Example 1

(b) Example 2

(c) Example 3

(d) Example 4

Figure 69: Examples of Blended Augmented Parent Images

Examples of the Blended Augmentation Training Set are shown above. Due to the way each GAN generated the exterior background information, it was very difficult to blend any of the images into the surrounding environment. As stated previously, there were more generated synthetic trucks and cars. Displayed in Figure 69d, the extra synthetic objects from these classes were used to create extremely diverse images uncaptured by the aerial vehicle in an attempt to enhance the YOLOv4-Tiny Model's generalizability.

VII. Follow-on Experiments' Results

Preamble

This chapter provides the results of the three different experiments conducted after the baseline analysis. The goal of Experiment I was to investigate the You Only Look Once Version 4 - Tiny (YOLOv4-Tiny) Model's generalizability on images of the same class but existing outside of the original population of images and observe if Generative Adversarial Network (GAN) augmentation bolstered model performance. In Experiment II, the goal was to implement hyperparameter tuning for the YOLOv4-Tiny Model in conjunction with GAN augmentation to gather evidence if using the two techniques together leads to better model performance. Finally, the goal of Experiment III was to use a multi-GAN "blended" augmentation technique to increase the diversity of an aerial image training set and produce the best performing detection model.

7.1 Experiment I: Alien Test Set

We hypothesized the minimal positive benefit of Generative Adversarial Networks (GANs) on the combined dataset was attributed to the nature of the test dataset. The test images were frames taken only moments after the training data images, providing minimal diversity. To understand the effects of a model's generalizability and robustness to new instances, an Alien Test Set was created with completely new and dissimilar instances. Experiment I was designed to determine if GAN augmentation could help bridge the gap of increasing an object detection model's ability to correctly localize and classify new instances of the same class, a difficult problem to solve in a computer vision.

Table 25: Experiment I Results: Learning Rate = 0.001

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	70.65%	0.0843	51.46%	0.70	0.64	0.67	183	78	103
Combined + DCGAN	**77.59%**	**0.1035**	**58.56%**	**0.78**	**0.60**	**0.60**	**171**	**47**	**115**
Combined + cDCAN	61.16%	0.1001	38.39%	0.53	0.52	0.53	150	133	136
Combined + WGAN	62.30%	0.0954	46.09%	0.62	0.51	0.56	145	88	141
Combined + cWGAN	68.40%	0.0930	50.22%	0.68	0.58	0.63	166	79	120

The only GAN augmented dataset supporting a better performing model than the model trained solely on the original images was the training set augmented with Deep Convolutional Generative Adversarial Network (DCGAN) images. The augmented training set containing DCGAN images was able to increase the model's Mean Average Precision (mAP) by 6.94% and the average Intersection over Union (IoU) accuracy by 7.10%. Precision also increased by 8.00%. The recall rate and F1-Score both decreased by 4.00% and 7.00%, respectively. The decline in True Positive (TP) detections and increase in False Negative (FN) detections led to the declination of the recall rate. The model failed at detecting 115 instances and falsely identified 47 instances. One inference from these results is the stark differences between the training set and test set led the model to fail in detecting the new object variants. For example, the trucks from the training set compared to the Alien Test Set are mostly similar in their size and shape but differ greatly in their physical attributes.

Table 26: Experiment I Results by Class

Dataset	Class	mAP	TP	FP
Combined	Van	74.22%	37	11
Combined + DCGAN	Van	83.65%	65	22
Combined	Truck	75.35%	68	10
Combined + DCGAN	Truck	74.87%	55	7
Combined	Car	62.39%	78	57
Combined + DCGAN	Car	74.23%	51	18

Even with the large amount of FN detections, performance was still improved on the detections of each of the classes when GAN augmented images were introduced

in training. For the van class, augmentation improved the mAP by 9.43% and added 28 TP detections. One negative impact was the 100.00% increase in False Positive (FP) detections which changed the ratio of the number of FP to TP detections from 29.73% to 33.85%. The truck class was the only class where we recorded a decline in mAP with a decrease of 0.48%. While TP detections also declined, the rate of FP to TP detections benefited from the GAN augmentation with a decrease of 1.98%. The car class had the highest net benefit from the augmented training dataset. The mAP increased by 11.84%. While TP detections fell, there was a dramatic decrease in FP detections which implies the model had less of a tendency to surmise a car was in a certain part of the frame in which it was not. The rate of FP to TP detections fell from 73.08 % to 35.29%, showing improvements in detection stability.

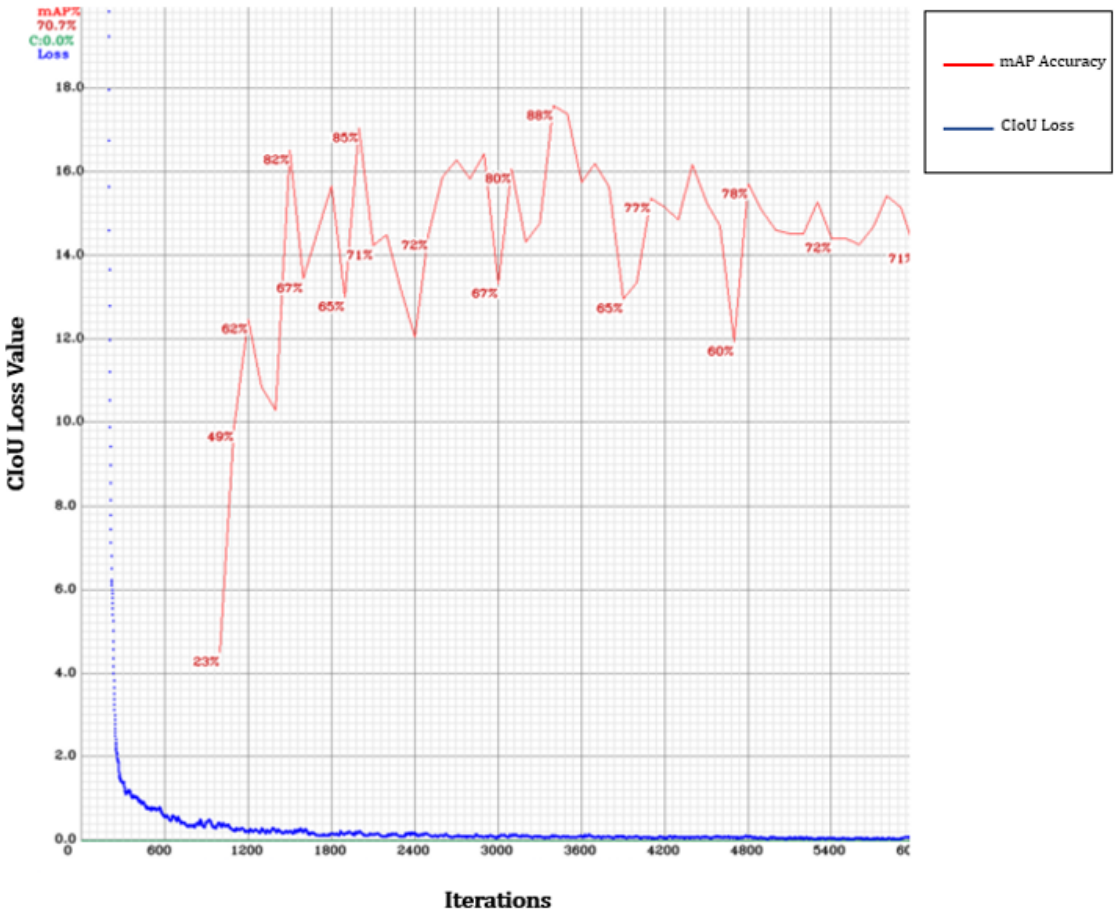


Figure 70: Experiment I: mAP vs. CIoU Loss: Combined Dataset + Alien Test Set

Both learning curves show a large variance in mAP during the training cycle, an effect occurring from the Alien Test Set. The augmented dataset achieved a higher mAP and shows to be more stable at the end of its training cycle compared to the unaltered dataset. GAN augmentation added more stability to training with test sets not deriving from the training set, an important feature for object detection models deployed in unknown environments to possess.

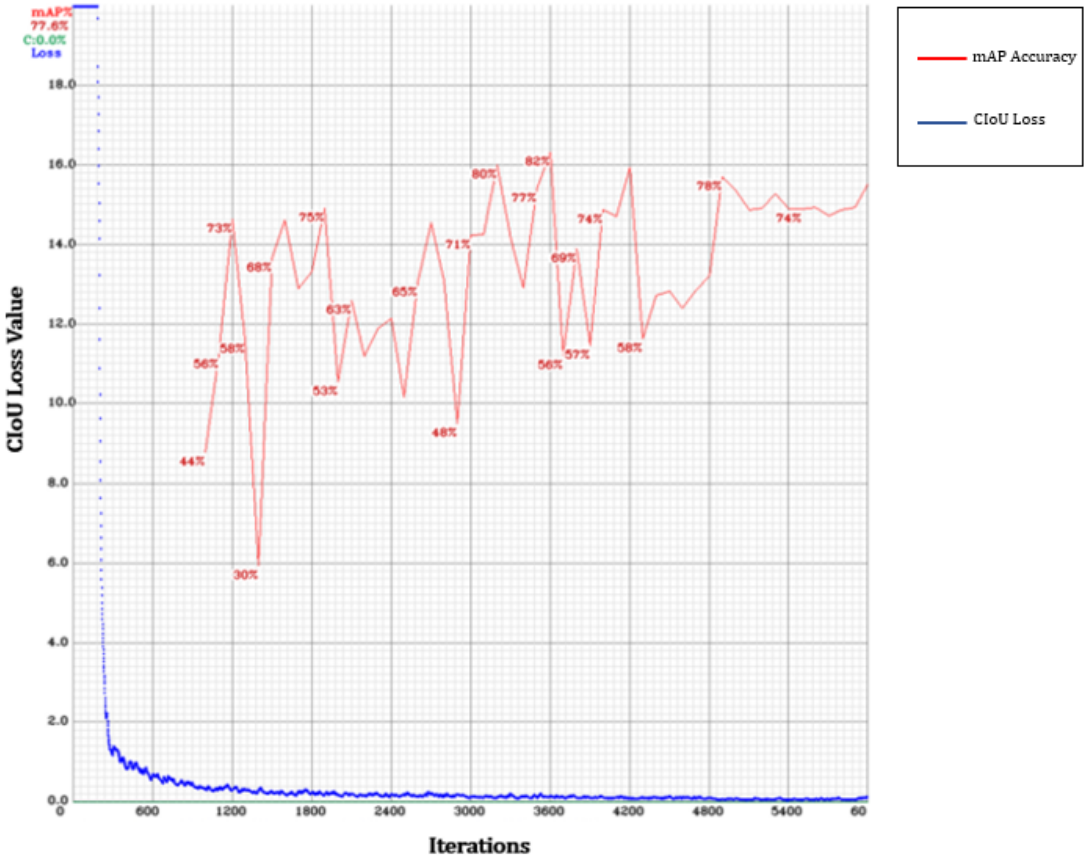


Figure 71: Experiment I: mAP vs. CIoU Loss: Combined Dataset + DCGAN + Alien Test Set

7.1.1 Discussion

An unexpected observation occurring in Experiment I was the least stable of the GANs, the DCGAN, supporting the only augmented dataset training a better performing model on the Alien Test Set compared to the original training set with no augmentation. One hypothesis for this observation is the DCGAN images were more diverse in their overall quality so the model was not bias towards the original test set and could perform better on the Alien Test Set. The other GANs trained with higher stability, generating images much more similar to the objects in the datasets. Further research should be conducted on augmented images with lower quality or

more dissimilar to their original counterparts to understand if their use will produce models more robust to detection on unique and diverse image datasets.

7.2 Experiment II: Alien Test Set with Varying Hyperparameters

After review of the results from Experiment I, we assert the DCGAN performed better than the other GANs for creating a robust model to the Alien Test Set due its lower quality and dissimilar images. This perhaps prevented the model from learning the original test set as well as the other augmented datasets, reducing its bias. Attempts to overcome the overfitting problem of the three other GAN variants involved changing the values of two different hyperparameters: the learning rate and number of iterations. Tuning these values forced the model to learn the objects derived from the original test set slower and generalize better to unrelated object instances.

The best model for both the original combined dataset and the GAN augmented dataset was attained from trials adopting a learning rate of 0.001305 and 4,000 epochs. The training set augmented with the Conditional Deep Convolutional Generative Adversarial Network (cDCGAN) images produced the most effective results. This model had the second highest accuracy but it had a much lower Complete Intersection over Union (CIoU) loss value than the model with the highest accuracy which was the model using a learning rate of 0.00261 and the training set augmented with images from the Conditional Wasserstein Generative Adversarial Network (cWGAN). The cWGAN augmented model had an accuracy of 78.49% but a CIoU loss value of 0.1820.

Table 27: Experiment II Results: Learning Rate = 0.001305 | Iterations = 4,000

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	74.94%	0.0847	52.38%	0.73	0.69	0.71	197	73	89
Combined + cDCGAN	76.52%	0.1312	53.92%	0.73	0.64	0.68	183	68	103

Using the Alien Test Set, the results show GANs were able to increase a model's generalizability on foreign images but future research is still needed to stabilize the localization of objects within a frame of an augmented dataset. The mAP was increased by 1.58% and the average IoU increased by 1.54%. A common pattern with the use of augmented objects was an increase in the CIoU loss, which increased by 4.65%. While there was no change in precision, the recall and F1-Scores fell by 5.00% and 3.00%, respectively. Performance degradation occurred with an increase of 14 FN detections from the combined dataset to the augmented combined dataset. FN detections recurrence may be connected to wider diversity of images provided by the GAN augmentation, leading to the model's confusion of where and what an object was in the frame. The TP detection value fell by 14 units also due to the model not completely learning where to look and accurately classify the objects in the Alien Test Set. The FP detection rate was positively affected in Experiment II with a decrease of 5 detections. The small decrease helped preserve the precision between the two models.

Table 28: Experiment II Results by Class: Learning Rate = 0.001305 | Iterations = 4,000

Dataset	**Class**	**mAP**	**TP**	**FP**
Combined	Van	76.18%	46	12
Combined + cDCGAN	Van	82.27%	37	0
Combined	Truck	78.42%	80	11
Combined + cDCGAN	Truck	74.47%	69	11
Combined	Car	60.81%	71	50
Combined + cDCGAN	Car	72.82%	77	57

While detection of the objects in Alien Test Set reaped huge benefits from GAN augmentation, better results were obtained for the classifications of the objects. The van object class had the best results for detection with a decrease of 12 to 0 FP detections. The mAP for the van class also increased by 6.09%. The truck was

the only object class to not gain any positive impacts from the augmented training set. A small decrease of 3.95% occurred in the mAP for the truck class. While TP detections fell by 11, the FP detection statistic did not change. The car object class received the most benefit from augmentation in the training set, gaining a 12.01% increase in mAP. The TP detections also increased for cars by 7, however, there was also an increase in the number of FP detections. More hyperparameter tuning should be researched to see how different hyperparameters in detection models benefit performance on datasets containing deepfake objects.

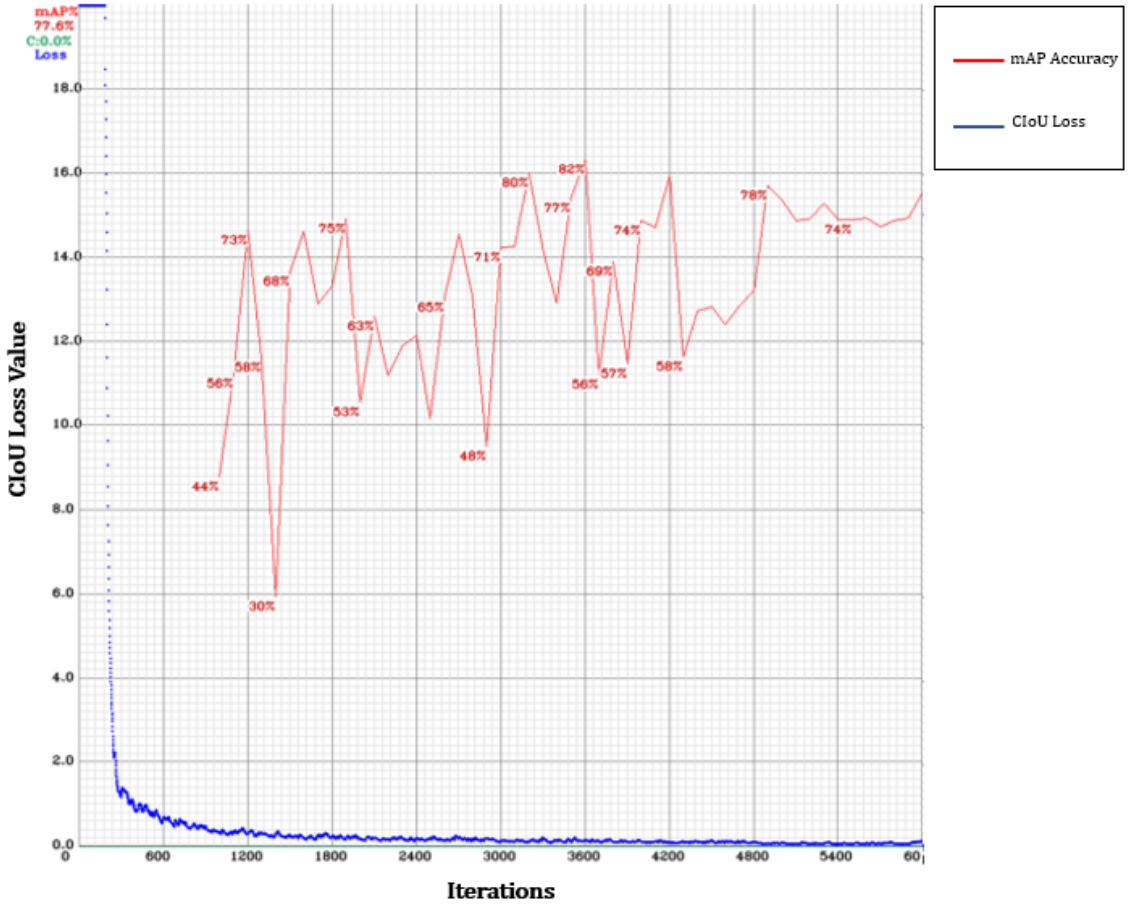


Figure 72: Experiment II: mAP vs. CIoU Loss: Combined Dataset + Alien Test Set

The mAP and CIoU graph for the model trained on the two different training sets showed similarities in the wide variance of the mAP over the training lifespan. Visual

inspection shows less volatility when training with the augmented dataset, providing evidence which shows GAN augmentation adds stability to the training process of the YOLOv4-Tiny Object Detection Model.

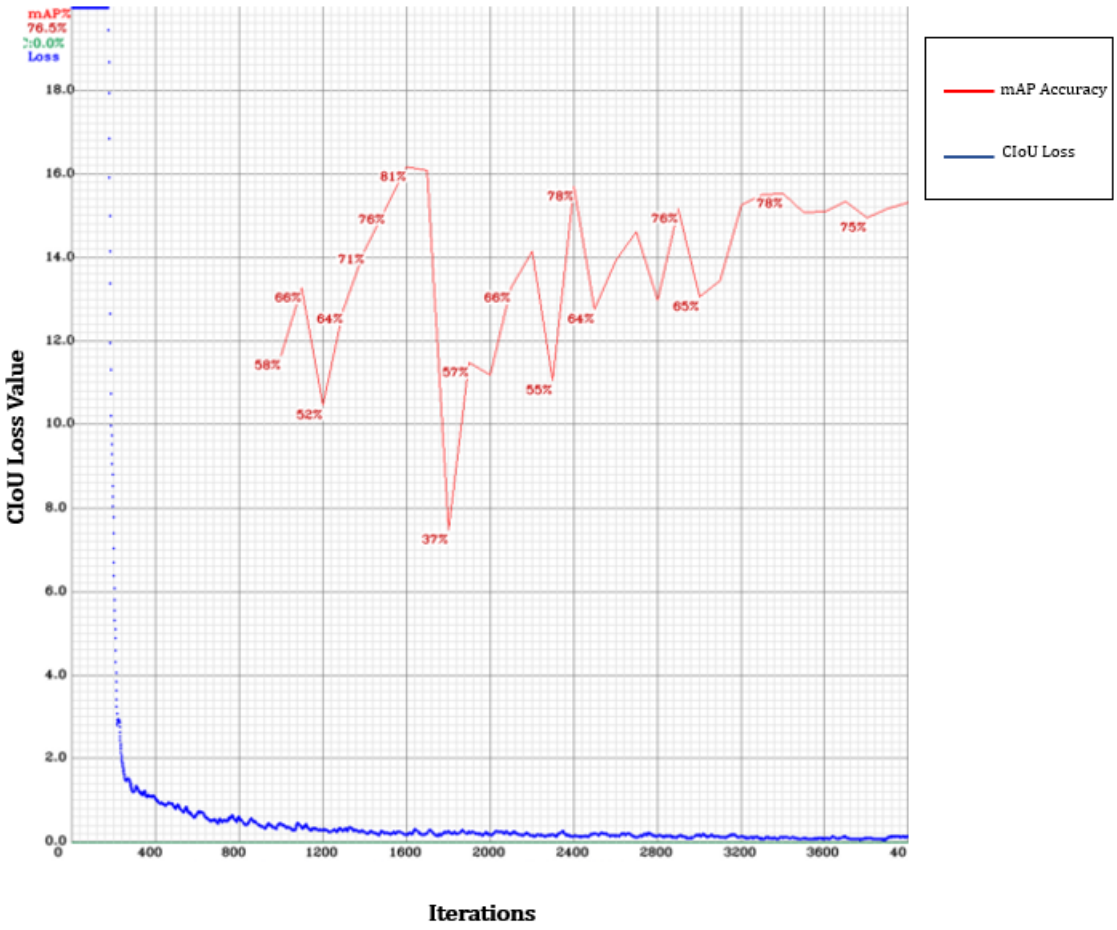


Figure 73: Experiment II: mAP vs. CIoU Loss: Combined Dataset + cDCGAN + Alien Test Set

7.2.1 Discussion

Experiment II showed the ability of aerial image training sets augmented with GANs to stabilize a model and provide higher levels of mAP on a test set which is completely foreign and different from the training set. These findings show great promise for solving the problem of CNNs classifying objects of the same class which

look completely different in physical nature. The use of a GAN helps add diversity and new features to a training set, promoting the performance of classification and detection models which can generalize to dramatically different, unknown instances.

Table 29: Experiment II Results: Performance Enhancements

Learning Rate	Iterations	%Δ in mAP	%Δ in IoU
0.00261	2000	13.08	13.34
0.00522	2000	2.12	-8.75
0.001305	2000	5.02	13.66

In Table 29, 2,000 iterations resulted in overall better performance in training the YOLOv4-Tiny Model compared to 4,000 iterations. This supports the hypothesis we made stating the model was overfitting when training for more iterations on the original dataset. All of the models gaining an increase in mAP also benefited from increases in average IoU besides the model with the learning rate 0.00522.

7.3 Experiment III: Blended Augmented Training Set

In Experiment III, we used a multi-GAN blended augmentation technique in an attempt to increase the diversity of the training set and performance of the model on both test sets. Three of the four models for the original training set produced mAP scores of 100.00%, showing the powerful effect the use of multiple GANs can have for increasing the range of diversity in a training set. Of the three best performing models, we selected the model with a learning rate of 0.00261 as the best model since it had the lowest CIoU loss value of 0.1060.

For the Alien Test Set, the model trained with a blended augmented dataset and learning rate of 0.001 achieved an accuracy of 85.67%. In comparison, this model achieved a mAP 7.18% higher than the highest accuracy achieved Experiment II.

That mAP was 78.49% with a cWGAN augmented training set using a learning rate of 0.00261.

Table 30: Experiment III Results

Test Set	Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Original	0.00261	100.00	0.1060	80.57%	0.96	1.00	0.98	170	8	0
Alien	0.001	85.67%	0.1464	60.24%	0.84	0.71	0.77	203	40	83

For the original test set, the blended augmentation technique engendered a powerful model with superior performance. Comparing the model to the training of the model using the combined training set with a learning rate of 0.00261, mAP increased by 2.25% to 100.00%. One negative impact occured in the loss metric increasing by 6.11%. A higher loss could be due to the increase in the amount of images. Training the model for more iterations could possibly have led to a reduction in the average loss value. Improvements were made in the average IoU value with an increase of 1.44%. Precision increase by 2.00%, and with zero FN detections, recall increased by 3.00% to 1.00. The value of the F1-Score also grew by 3.00% to 0.98. TP detections grew by a value of 65 to 170 and FP detections decreased by 3 to a value of 8. Further support for the model's robustness and detection ability is the improvement from 5 FN detections to zero FN detections, providing proof the model is more capable of finding the correct foreign objects in the frame.

Using the blended augmentation technique for the original training set provided a similar pattern of improvements in the model's performance when compared against the Alien Test Set. Comparing the Blended Augmentation Training Set model to the model using the original combined training set and the Alien Test Set, mAP improved by 15.02%. The average IoU rate increased from 51.46% to 60.24%. Consistent with the other training trial in Experiment II, CIoU loss increased by 6.21%. For all model comparisons, the Blended Augmentation Training Set led to the biggest collective improvements on models using the Alien Test Set for the precision, recall, and F1-

Score rates. Precision was increased by 14.00%, recall had improvements of 7.00%, and the F1-Score was improved by 10.00%. The TP detections increased by 20 and FN detections decreased by 20. FP detections had the biggest decrease of 48 detections changing from the original value of 78 to 40. The shifts in positive directions for all the performance metrics provides evidence supporting GAN augmentation in training sets for creating and enhancing viable models for object detection in unknown and different environments.

Table 31: Experiment III Results by Class: Blended Augmentation | Original Test Set

Class	**mAP**	**TP**	**FP**
Van	100.00%	48	0
Truck	100.00%	75	6
Car	100.00%	47	2

Investigating each detection class also provides support for the assertion stating the blended augmentation technique is beneficial for augmenting datasets used for training the YOLOv4-Tiny Model. First, comparing the two models trained with the original test set, the mAP for the van detections increased by 4.86%. The number of TP detections increased by 5 and the value of FP detections stayed the same at 0. For the truck class, mAP increased by 0.08% to 100.00%. TP detections stayed the same while FP detections increased by 1. There was little impact on the truck class, however, it was already at the upper threshold of feasible rates before augmentation. One reason there was no reduction in the FP detection rate of trucks was the class displayed the highest amount of complexity in its features for the GAN to recreate and model to learn. Finally, the car class had a 1.83% increase in its mAP value. TP car detections increased by 3 to 47 and the biggest drop in the FP rate occured for the car with a total decrease of 9 units.

Table 32: Experiment III Results by Class: Blended Augmentation | Alien Test Set

Class	**mAP**	**TP**	**FP**
Van	90.09%	53	2
Truck	75.53%	77	21
Car	91.41%	73	17

All three objects of interest had improvements from the Blended Augmentation Training Set and Alien Test Set implementation. mAP for the van class improved by 15.87% with 16 more TP detections and 9 less FP detections. The truck class gained very small improvements, receiving an improvement in its mAP score of 0.18%. The number of TP detections positively increased by 9 but was offset by an increase of 11 detections for the FP detection metric. One of the reasons it was hard to train the YOLOv4-Tiny Model to correctly find and classify the truck class was the three types of trucks captured by Unmanned Aerial Vehicles (UAVs) between the two sets were dramatically different in their physical appearance. Finally, the car class benefited from a mAP increase of 29.02% which was the biggest improvement for any classes' mAP through all trials and experiments. While the mAP increased, TP detections fell by 5, but this declination was offset by a huge improvement in the FP detections which decreased by 40 units.

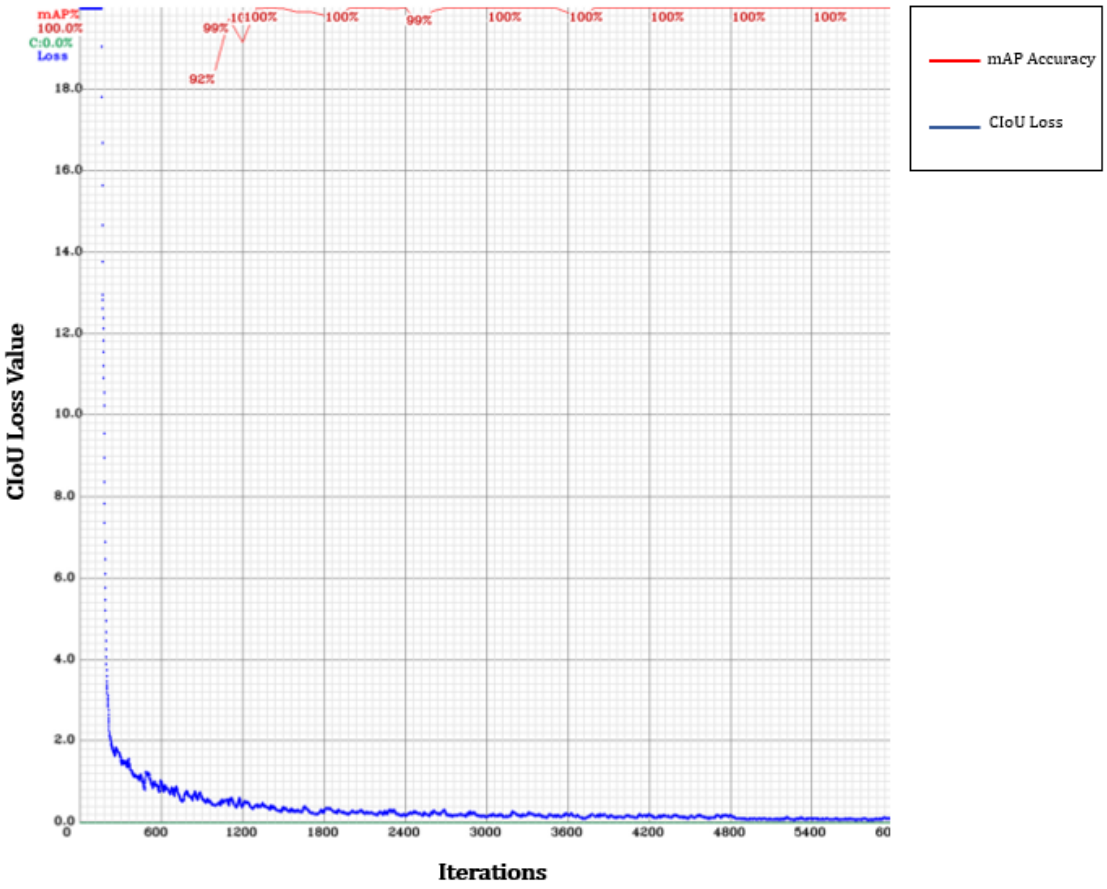


Figure 74: Experiment III: mAP vs. CIoU Loss Curve: Original Test Set

For the original test set, the Blended Augmentation Training Set supported the model in reaching a 100.00% mAP in approximately 1,400 iterations. mAP was stable afterwords with a constant decreasing CIoU loss, leading to the conclusion asserting the Blended Augmentation Training Set helped bolster a strong, balanced object detection model.

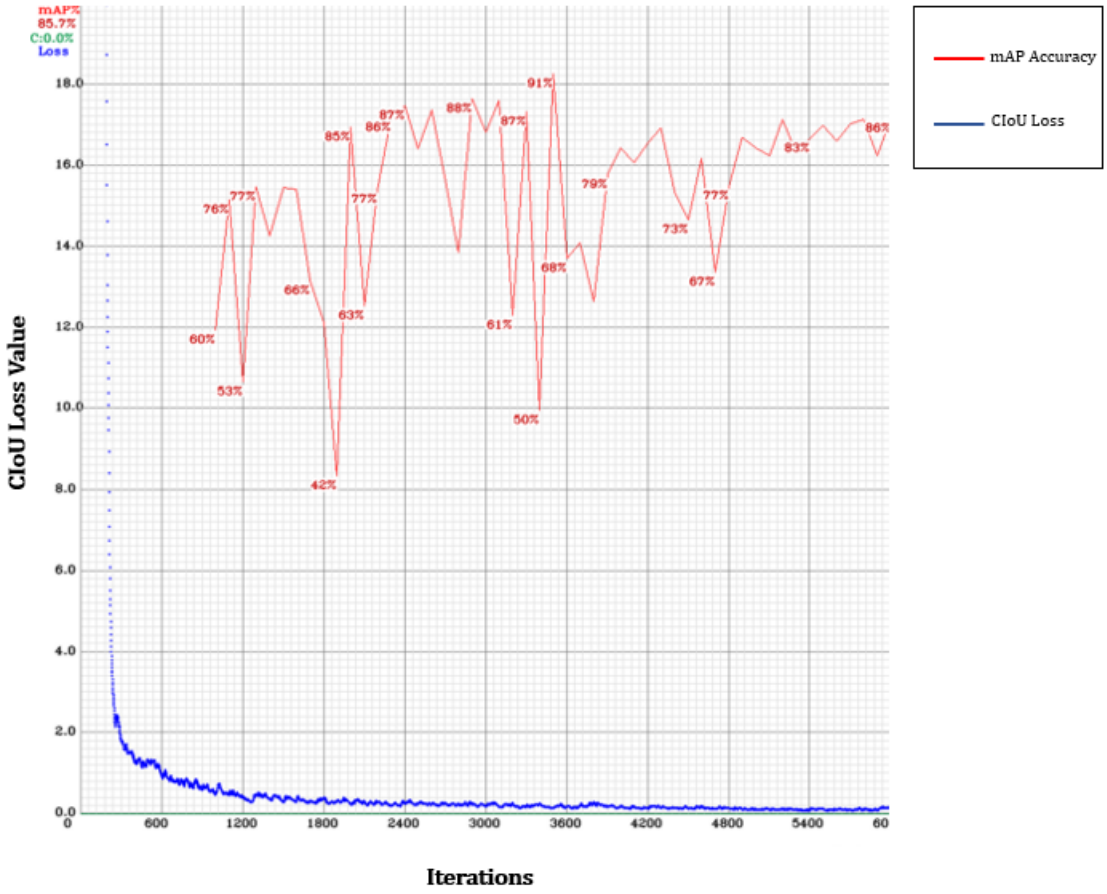


Figure 75: Experiment III: mAP vs. CIoU Loss Curve: Alien Test Set

The mAP for the Blended Augmentation Training Set with the Alien Test Set had a wide variance over the 6,000 iterations due to the huge differences between the objects in two sets. The model's mAP plateaued at 4,000 iterations and began displaying signs of a balanced object detection model. Further research is recommended to analyze the use of GANs for augmenting objects in a training set which differ from the objects of the same class in a test set on what hyperparameters need to be tuned to allow for more stability in the model training.

7.3.1 Discussion

Our main takeaway from Experiment III is augmenting a training set with an enormous amount of different GAN instances will lead to strong performing detection models. The use of the blended augmentation on the training set confirmed the assertion using GANs for augmenting aerial image training sets will lead to more effective localization and classification capabilities for object detection models. The average of the mAP value for the models using the Blended Augmentation Training Set and Original Test Set was 98.99%. For the models using the Blended Augmentation Training Set and Alien Test Set, mAP averaged at a rate of 73.71%. An important finding from Experiment III is the ability to use GANs to teach models to find objects which are of the same class but differ in qualitative features with acceptable accuracy and detection rates. The implementations of UAVs in unknown environments will benefit from the use of GAN augmentation in their models' training sets to increase their detection performance.

VIII. Conclusions

Generative Adversarial Network (GAN) augmentation for enlarging the size of image training sets acquired by aerial moving platforms showed effectiveness in increasing the accuracy of object detection models. With a 6.75% increase in Mean Average Precision (mAP) and a best-case increase of 15.76%, GAN augmentation proved to provide quality, brand new training instances not captured by a moving platform for the model, expanding the model's generalizability and robustness to new images. Similarly, we saw a 4.13% increase on average and a best-case increase of 9.60% for the Intersection over Union (IoU), providing further evidence for the assertion stating GAN augmentation will better support object detection models. The use of synthetic images from GANs in aerial image training sets decreases the amount of time and manpower data scientists must spend on data collection and can create real-time deployable detection models in a shorter time frame. The only weakness of GAN augmentation for training aerial image object detection models was the increase in the Complete Intersection over Union (CIoU) loss metric, however, pragmatically the substantial increases in both mAP and IoU offset this problem and yielded much better preforming models. Further analysis is recommended to discover what parameters were having an effect on the increase in CIoU loss and if any CIoU loss mitigation can be obtained from hyperparameter tuning.

A current problem faced in training machine learning (ML) models, especially Convolutional Neural Networks (CNNs), is training the model to be able to recognize objects of the same class but with different physical features. Using the Alien Test Set, Generative Adversarial Networks (GANs) showed great potential in bridging the gap of vacuity surrounding new instances of objects with the addition of unique, diverse characteristics in the synthetic images produced from a GAN. When using the blended augmentation technique for creating new training sets, the highest mAP value

of 85.67% was achieved on the Alien Test Set. This was a 15.02% increase compared to the You Only Look Once Version 4 - Tiny (YOLOv4-Tiny) Model trained on the original combined training set with no augmentation using a learning rate of 0.001 and 6,000 epochs. Moreover, using the Blended Augmentation Training Set in training led to an 8.78% improvement in the average IoU rate for the YOLOv4-Tiny Object Detection Model, increasing average IoU from 51.46% to 60.24%.

Previously outlined in section 1.1.1, the research sought to answer three questions:

1. Does an augmented image training dataset acquired by a moving platform containing generated synthetic images from a GAN increase the classification accuracy and generalizability of a Convolutional Neural Network (CNN) object detection model?

2. Does an augmented image training dataset acquired by a moving platform containing generated synthetic images from a GAN increase the localization and generalizability of a CNN object detection model?

3. What inferences can be drawn from the unaugmented and augmented datasets that show their similarities and dissimilarities?

The first research question proposed was seeking to illicit proof of whether data augmentation from GANs can increase the accuracy of a CNN model used for object detection. Improvements were discovered in mAP of 6.75% on average when using GAN augmentation. Additional research may want to compare GAN augmentation to other augmentation techniques on aerial image training sets to discover which type of technique produces the best model mAP. The second research question to be answered was whether the localization of an object detection model could be increased with a GAN data augmentation technique. The 4.13% average improvement of the average IoU rate showed the model was able to properly locate the object of interest

within a frame. Further research is recommended to understand how a model could detect objects of the same class but dramatically different in their sizes.

Questions one and two both sought to investigate generalizability enhancement of an object detection model for an aerial vehicle. Experiments I, II, and III were designed using an Alien Test Set to provide evidence in support of GAN augmentation for generalizability enhancement. GAN augmentation showed to be most effective in Experiments II and III. In Experiment I, the use of too many samples similar to the original dataset hindered the generalizability of the model and overfit the model to the wrong test set. This is one consideration to be taken into account when augmenting training datasets with GANs which are training models used for real world applications. Diversity in the training set images is a prerequisite for creating a suitable model for deployment in a real-world scenario.

Finally, the last research question posed wanted to provide justification for the assertion stating the quality of the generated images aligned with the original images found in the provided datasets. This was done through human inspection and verification of each synthetic image. One future research area for GANs is finding a quick process for verifying the quality of generated images. Analysis of the results also helped clarify the quality of the images. Increases in CIoU loss may allude to lower quality synthetic images, but this can also be offset with increases in the accuracy, precision, and recall for a ML model. The YOLOv4-Tiny Model followed the pattern of small increases in CIoU loss but large increases in the accuracy metrics of the model, suggesting the synthetic images produced were of higher quality. Quality assessment of data generated by GANs is still in the infancy stage of this ML technique and should be pursued to benefit not only augmented images from moving aerial platforms but other types of data as well.

8.1 Future Research

The research presented in this thesis showed the feasibility and effectiveness of augmenting moving platform aerial image training sets with GANs but there is further investigations which should be pursued to increase the understanding of GANs implementation and strengthening the generalizability of object detection models. Future research endeavors include:

- Implementation of a Progressive Growing Generative Adversarial Network (PGGAN) or a StyleGAN to augment the entire frame.
- Replacement of Rectified Linear Unit (ReLU) and Leaky Rectified Linear Unit (LReLU) activation functions in GAN architectures to investigate the impacts on stability.
- Investigation of whether specific angles of salient objects in a frame can be mapped to in a vector space by a GAN, as well as other native features of aerial videos such as variations in altitude.
- Research on finding quick ways of verifying the quality of synthetic images produced by GANs as well as a concrete way of confirming the training cycle has reached its limit of creating new images.

The next adaptation of the presented research would be to use a PGGAN or StyleGAN to create augmented images of the entire original frames captured by the Unmanned Aerial Vehicle (UAV). A PGGAN will provide the capabilities of recreating the originally collected images which were greater than size 64x64. PGGANs have shown the ability to create high quality images of synthetic faces [76] and could allow for the development of producing images which are similar in quality to the type of camera used to capture images on various different moving platforms today. This

could lead to development of new types of images which include not only new diverse objects of interest but new diverse external backgrounds with a larger image size and level of image quality, producing a more generalized object detection model. The use of a StyleGAN allows for control over the different details of the generated images using different style vectors and noise. The three major changes to the StyleGAN occur in the generator [76]. The first change is mapping the points in the latent space to an intermediate latent space through a mapping network. The second change stems off of the first change since now the intermediate latent space can be used to manage the generated output. The third change utilizes noise as the origin for variation at each point in the generator. Mapping different characteristics of aerial images to style vectors can allow for new training images to be created which help balance not only the number of objects in the dataset, but the types of images located within in the dataset. For example, mapping altitudes to style vectors could allow for the controlled generation of synthetic aerial images at various altitudes, from 10,000 feet to 20,000 feet.

While the ReLU and LReLU are standard activation functions currently used today, the problem of vanishing and exploding gradients is still prevalent, especially in training GANs. Network stability is one of the main problems faced when implementing GANs and finding further methods or architectural changes which can lead to a higher amount of stability in a GAN can generate higher quality and more realistic images, producing a more robust model capable of real-time detections. Two recommended activation functions to investigate in GAN architecture which have been shown in computer vision research to produce more stability in CNN model training are the Mish and Elliott Activation Functions.

Understanding how a GAN interacts with aerial data can be beneficial for creating new, diverse images varying in their main characteristics occurring from imagery

captured by continuously moving objects. Two characteristics of interest are the different angles of an object which the moving platform detects and captures the object at as well as the altitude the aircraft is at the time of detection. Developing a way to map the different instance angles to a vector in the latent space could allow for a GAN to be conditioned with the captured angles and produce different vehicles at various angle degrees. Angle conditioning of a GAN will provide new ways of generating an object seen from a different spot relative to platform originally uncaptured. This could help bolster the detection capabilities of an object detection model as it moves through an area of interest and captures objects at various different times and locations. Another characteristic of film acquired from aerial moving platforms is the altitude of the aircraft at the time the frame was obtained. The ability to produce new instances at different altitude levels will decrease the amount of data collection needed by different air crafts at those altitudes and allow for deployable detection models in a shorter time frame.

Appendix A. Appendix

1.1 GAN Python Code Examples

1.1.1 cDCGAN Python Code

```
#Packages
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
```

```
#Set-up
class Discriminator(nn.Module):
  def __init__(self, img_channels, features_disc, num_classes,
                                        image_size):
    super(Discriminator, self).__init__()
    self.image_size = image_size
    self.disc = nn.Sequential(
        nn.Conv2d(
            img_channels + 1, features_disc, kernel_size=4, stride=2
                                                , padding=1
        ),
        nn.LeakyReLU(0.2),
    self._block(features_d, features_disc * 2, 4, 2, 1),
            self._block(features_disc * 2, features_disc * 4, 4, 2,
                                                1),
            self._block(features_disc * 4, features_disc * 8, 4, 2,
                                                1),
```

```
            nn.Conv2d(features_disc * 8, 1, kernel_size=4, stride=2,
                                                    padding=0),
            nn.Sigmoid(),
        )
    self.embed = nn.Embedding(num_classes,image_size*image_size)

  def _block(self, in_channels, out_channels, kernel_size, stride,
                                      padding):
    return nn.Sequential(
            nn.Conv2d(
                in_channels,
                out_channels,
                kernel_size,
                stride,
                padding,
                bias=False,
            ),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(0.2),
        )

  def forward(self, x, labels):
    embedding = self.embed(labels).view(labels.shape[0],1,self.
                                        image_size, self.image_size)
    x = torch.cat([x,embedding],dim=1)
    return self.disc(x)


class Generator(nn.Module):
  def __init__(self, channels_noise, img_channels, features_gen,
                                      num_classes, image_size,
                                      embed_size):
```

```
    super(Generator, self).__init__()
    self.image_size = image_size
    self.net = nn.Sequential(
        self._block(channels_noise + embed_size, features_gen * 16,
                                            4, 1, 0),  # img: 4x4
        self._block(features_gen * 16, features_gen * 8, 4, 2, 1),
                                            # img: 8x8
        self._block(features_gen * 8, features_gen * 4, 4, 2, 1),  #
                                             img: 16x16
        self._block(features_gen * 4, features_gen * 2, 4, 2, 1),  #
                                             img: 32x32
        nn.ConvTranspose2d(
            features_gen * 2, img_channels, kernel_size=4, stride=2,
                                                  padding=1
        ),
        nn.Tanh(),
        )
    self.embed = nn.Embedding(num_classes, embed_size)

def _block(self, in_channels, out_channels, kernel_size, stride,
                                       padding):
    return nn.Sequential(
            nn.ConvTranspose2d(
                in_channels,
                out_channels,
                kernel_size,
                stride,
                padding,
                bias=False,
            ),
            nn.ReLU(),
        )
```

```
  def forward(self, x, labels):
    embedding = self.embed(labels).unsqueeze(2).unsqueeze(3)
    x = torch.cat([x, embedding], dim=1)
    return self.net(x)


def initialize_weights(model):
  for m in model.modules():
    if isinstance(m, (nn.Conv2d, nn.ConvTranspose2d, nn.BatchNorm2d)
                                          ):
      nn.init.normal_(m.weight.data, 0.0, 0.02)
```

```
#Training
#For this implementation, the directory should have a class labeled
#folder for each object
dataset = datasets.ImageFolder(root="FILENAME", transform=transforms
                                      )
dataloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=True
                                      )

device = torch.device("cuda" if torch.cuda.is_available() else "cpu"
                                      )
LEARNING_RATE = #DEFINE
BATCH_SIZE = #DEFINE
IMG_SIZE = #DEFINE
IMG_CHANNELS = #DEFINE
NOISE_DIMENSIONS = #DEFINE
EPOCHS = #DEFINE
FEATURES_DISC = #DEFINE
FEATURES_GEN = #DEFINE
CLASSES = 4
```

```
GEN_EMBEDDING = 100

transforms = transforms.Compose(
    [
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize(
            [0.5 for _ in range(IMG_CHANNELS)], [0.5 for _ in range(
                                                    IMG_CHANNELS)]
        ),
    ]
)



gen = Generator(NOISE_DIMENSIONS,
IMG_CHANNELS,
FEATURES_GEN,
CLASSES,
IMG_SIZE,
GEN_EMBEDDING).to(device)
disc = Discriminator(IMG_CHANNELS, FEATURES_DISC, CLASSES, IMG_SIZE)
                                    .to(device)
initialize_weights(gen)
initialize_weights(disc)

opt_gen = optim.Adam(gen.parameters(), lr=LEARNING_RATE, betas=(0.5,
                                     0.999))
opt_disc = optim.Adam(disc.parameters(), lr=LEARNING_RATE, betas=(0.
                                     5, 0.999))
criterion = nn.BCELoss()
```

```
#Putting them into training mode
gen.train()
disc.train()

for EPOCH in range(EPOCHS):
  for batch_idx, (real, labels) in enumerate(dataloader):
    real = real.to(device)
    labels = labels.to(device)
    current_batch_size = real.shape[0]
    noise = torch.randn(current_batch_size, NOISE_DIMENSIONS, 1, 1).
                                        to(device)
    fake = gen(noise, labels)

    #Discriminator Training
    disc_real = disc(real, labels).reshape(-1)
    loss_disc_real = criterion(disc_real, torch.ones_like(disc_real)
                                        ) #loss discriminator real
                                        images
    disc_fake = disc(fake.detach(), labels.detach()).reshape(-1) #
                                        loss discriminator fake images
    loss_disc_fake = criterion(disc_fake, torch.zeros_like(disc_fake
                                        )) #Loss on the fake images
    loss_disc = (loss_disc_real + loss_disc_fake) / 2
    disc.zero_grad()
      #loss_disc.backward()
    loss_disc.backward(retain_graph=True)
    opt_disc.step()

    #Generator Training
    output = disc(fake, labels).reshape(-1)
    loss_gen = criterion(output, torch.ones_like(output))
    gen.zero_grad()
```

```
    loss_gen.backward()
    opt_gen.step()
```

1.1.2 cWGAN Python Code

```
class Discriminator(nn.Module):
  def __init__(self, img_channels, features_disc, num_classes,
                                        image_size):
    super(Discriminator, self).__init__()
    self.image_size = image_size
    self.disc = nn.Sequential(
        nn.Conv2d(
            img_channels + 1,
            features_disc,
            kernel_size=4,
            stride=2,
            padding=1
        ),
        nn.LeakyReLU(0.2),
    self._block(features_disc, features_disc * 2, 4, 2, 1),
            self._block(features_disc * 2, features_disc * 4, 4, 2,
                                                1),
            self._block(features_disc * 4, features_disc * 8, 4, 2,
                                                1),
            nn.Conv2d(features_disc * 8, 1, kernel_size=4, stride=2,
                                                 padding=0),
        )
    self.embed = nn.Embedding(num_classes, image_size * image_size)

  def _block(self, in_channels, out_channels, kernel_size, stride,
                                        padding):
    return nn.Sequential(
```

```
            nn.Conv2d(
                in_channels,
                out_channels,
                kernel_size,
                stride,
                padding,
                bias=False,
            ),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(0.2),
        )

    def forward(self, x, labels):
        embedding = self.embed(labels).view(labels.shape[0], 1, self.
                                            image_size, self.image_size)
        x = torch.cat([x, embedding], dim=1)
        return self.disc(x)


class Generator(nn.Module):
    def __init__(self, channels_noise, img_channels, features_g,
                 num_classes,
                 image_size,
                 embed_size):
        super(Generator, self).__init__()
        self.image_size = image_size
        self.net = nn.Sequential(
            self._block(channels_noise + embed_size, features_gen * 16,
                                                4, 1, 0),  # img: 4x4
            self._block(features_gen * 16, features_gen * 8, 4, 2, 1),
                                                # img: 8x8
```

```
        self._block(features_gen * 8, features_gen * 4, 4, 2, 1),  #
                                                    img: 16x16
        self._block(features_gen * 4, features_gen * 2, 4, 2, 1),  #
                                                    img: 32x32
        nn.ConvTranspose2d(
            features_gen * 2, img_channels, kernel_size=4, stride=2,
                                                        padding=1
        ),
        nn.Tanh(),
        )
    self.embed = nn.Embedding(num_classes, embed_size)

def _block(self, in_channels, out_channels, kernel_size, stride,
                                        padding):
    return nn.Sequential(
            nn.ConvTranspose2d(
                in_channels,
                out_channels,
                kernel_size,
                stride,
                padding,
                bias=False,
            ),
            nn.ReLU(),
        )

def forward(self, x, labels):
    embedding = self.embed(labels).unsqueeze(2).unsqueeze(3)
    x = torch.cat([x,embedding], dim=1)
    return self.net(x)
```

```
def initialize_weights(model):
  for m in model.modules():
    if isinstance(m, (nn.Conv2d, nn.ConvTranspose2d, nn.BatchNorm2d)
                                        ):
      nn.init.normal_(m.weight.data, 0.0, 0.02)
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader


# Hyperparameters
device = torch.device("cuda" if torch.cuda.is_available() else "cpu"
                                      )
LEARNING_RATE = #DEFINE
BATCH_SIZE = #DEFINE
IMG_SIZE = #DEFINE
IMG_CHANNELS = #DEFINE
NOISE_DIMENSIONS = #DEFINE
EPOCHS = #DEFINE
FEATURES_DISC = #DEFINE
FEATURES_GEN = #DEFINE
CRITIC_ITERATIONS = #DEFINE
CLIPPING_WEIGHT = #DEFINE
CLASSES = #DEFINE
GEN_EMBEDDING = #DEFINE


transforms = transforms.Compose(
    [
```

```
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize(
            [0.5 for _ in range(IMG_CHANNELS)], [0.5 for _ in range(
                                                IMG_CHANNELS)]
        ),
    ]
)


dataset = datasets.ImageFolder(root=FILENAME, transform=transforms)

dataloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=True
                                    )
gen = Generator(NOISE_DIMENSIONS, IMG_CHANNELS, FEATURES_GEN,
                                    CLASSES, IMG_SIZE, GEN_EMBEDDING).
                                    to(device)
critic = Discriminator(IMG_CHANNELS, FEATURES_DISC, CLASSES,
                                    IMG_SIZE).to(device)
initialize_weights(gen)
initialize_weights(critic)

opt_gen = optim.RMSprop(gen.parameters(), lr=LEARNING_RATE)
opt_critic = optim.RMSprop(critic.parameters(), lr=LEARNING_RATE)

#Training Mode
gen.train()
critic.train()

for EPOCH in range(EPOCHS):
  for batch_idx, (real, labels) in enumerate(dataloader):
    real = real.to(device)
```

```
    labels = labels.to(device)
    current_batch_size = real.shape[0]

    for i in range(CRITIC_ITERATIONS):
      noise = torch.randn(current_batch_size, NOISE_DIMENSIONS, 1, 1
                                          ).to(device)
      fake = gen(noise, labels)
      critic_real = critic(real, labels).reshape(-1)
      critic_fake = critic(fake, labels).reshape(-1)
      loss_critic = -(torch.mean(critic_real) - torch.mean(
                                          critic_fake)) #Expected
                                          Value of Critic on Real -
                                          Expected Value of Critic on
                                          Fake
      critic.zero_grad()
      loss_critic.backward(retain_graph=True)
      opt_critic.step()

      for p in critic.parameters():
        p.data.clamp_(-CLIPPING_WEIGHT, CLIPPING_WEIGHT)

    output = critic(fake, labels).reshape(-1)
    loss_gen = -torch.mean(output)
    gen.zero_grad()
    loss_gen.backward()
    opt_gen.step()
```

1.2 YOLOv4-Tiny Hyperparameter Values

Table 33: YOLOv4-Tiny Model Hyperparameter Values

Hyperparameter	**Value**
Batch Size	64
Subdivisions	16
Image Width	416
Image Height	416
Color Channels	3
Momentum	0.90
Decay	0.0005
Saturation	1.5
Exposure	1.5
Hue	0.1
Learning Rate	0.00261-0.00001
Policy	Steps
Steps	4800,5400

1.3 Training Results: Dataset 1

Table 34: YOLOv4-Tiny Model Training Results: Dataset 1

Learning Rate	**mAP**	**CIoU Loss**	**Avg. IoU**	**Precision**	**Recall**	**F1-Score**	**TP**	**FP**	**FN**
0.00261	84.18%	0.0321	57.72%	0.72	0.81	0.76	137	52	33
0.001	**88.19%**	**0.040**	**64.34%**	**0.81**	**0.82**	**0.82**	**140**	**33**	**30**
0.0001	82.47%	0.0994	57.90%	0.76	0.78	0.77	133	43	37
0.00001	66.51%	0.2943	56.59%	0.83	0.48	0.61	82	17	88

Table 36: YOLOv4-Tiny Model Training Results: Dataset 1 + DCGAN Images

Learning Rate	**mAP**	**CIoU Loss**	**Avg. IoU**	**Precision**	**Recall**	**F1-Score**	**TP**	**FP**	**FN**
0.00261	86.45%	0.0371	65.26%	0.82	0.79	0.80	134	29	36
0.001	84.73%	0.0477	65.88%	0.82	0.76	0.79	130	28	40
0.0001	90.55%	0.1044	65.36%	0.86	0.81	0.83	138	23	32
0.00001	75.20%	0.3106	58.86%	0.83	0.70	0.61	103	21	67

Table 35: YOLOv4-Tiny Model Training Results by Class: Dataset 1

Class	Learning Rate	mAP	TP	FP
Van	0.001	99.67%	48	9
Van	0.00261	99.17%	48	23
Van	0.0001	96.39%	46	6
Van	0.00001	38.45%	4	2
Truck	0.001	65.26%	45	22
Truck	0.00261	60.64%	45	18
Truck	0.0001	73.41%	52	23
Truck	0.00001	69.82%	35	3
Car	0.001	99.60%	47	2
Car	0.00261	92.73%	44	11
Car	0.0001	77.62%	35	14
Car	0.00001	91.25%	43	12

Table 37: YOLOv4-Tiny Model Training Results by Class: Dataset 1 +DCGAN Images

Class	Learning Rate	mAP	TP	FP
Van	0.001	99.30%	47	5
Van	0.00261	99.20%	47	12
Van	0.0001	96.53%	46	6
Van	0.00001	88.57%	41	15
Truck	0.001	68.18%	44	15
Truck	0.00261	69.22%	48	11
Truck	0.0001	78.24%	53	16
Truck	0.00001	46.86%	19	0
Car	0.001	86.72%	39	8
Car	0.00261	90.93%	39	6
Car	0.0001	96.89%	39	1
Car	0.00001	90.17%	43	6

Table 38: YOLOv4-Tiny Model Training Results: Dataset 1 + cDCGAN Images

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	90.63%	0.0489	61.60%	0.74	0.79	0.77	134	46	36
0.001	91.83%	0.0632	69.25%	0.86	0.83	0.84	141	23	29
0.0001	94.25%	0.1219	65.49%	0.86	0.91	0.89	155	25	15
0.00001	78.47%	0.3098	54.18%	0.76	0.62	0.68 1	105	34	65

Table 39: YOLOv4-Tiny Model Training Results by Class: Dataset 1 + cDCGAN Images

Class	Learning Rate	mAP	TP	FP
Van	0.001	98.51%	47	5
Van	0.00261	99.53%	47	13
Van	0.0001	98.11%	47	4
Van	0.00001	82.53%	42	29
Truck	0.001	76.96%	47	8
Truck	0.00261	74.43%	41	4
Truck	0.0001	85.55%	63	16
Truck	0.00001	61.40%	20	1
Car	0.001	100.00%	47	10
Car	0.00261	97.93%	46	29
Car	0.0001	99.10%	45	5
Car	0.00001	91.47%	43	4

Table 40: YOLOv4-Tiny Model Training Results: Dataset 1 + WGAN Images

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	88.44%	0.0374	65.12%	0.79	0.79	0.79	135	36	35
0.001	90.95%	0.0518	66.13%	0.81	0.81	0.81	138	32	32
0.0001	90.00%	0.1156	59.66%	0.79	0.84	0.81	142	38	28
0.00001	76.31%	0.3111	46.38%	0.67	0.49	0.57	84	42	86

Table 41: YOLOv4-Tiny Model Training Results by Class: Dataset 1 +WGAN Images

Class	Learning Rate	mAP	TP	FP
Van	0.001	99.64%	48	19
Van	0.00261	98.98%	47	28
Van	0.0001	95.04%	46	19
Van	0.00001	86.82%	44	38
Truck	0.001	73.24%	43	11
Truck	0.00261	66.56%	41	3
Truck	0.0001	75.00%	52	19
Truck	0.00001	54.24%	0	0
Car	0.001	99.96%	47	2
Car	0.00261	99.78%	47	5
Car	0.0001	99.96%	44	0
Car	0.00001	87.86%	40	4

Table 42: YOLOv4 Model-Tiny Training Results: Dataset 1 + cWGAN Images

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	90.24%	0.0349	62.51%	0.76	0.79	0.83	141	44	29
0.001	88.61%	0.0525	62.79%	0.77	0.81	0.80	136	41	34
0.0001	94.78%	0.1223	67.24%	0.86	0.84	0.85	145	24	25
0.00001	80.14%	0.3232	55.58%	0.82	0.49	0.61	103	22	67

Table 43: YOLOv4-Tiny Model Training Results by Class: Dataset 1 +cWGAN Images

Class	Learning Rate	mAP	TP	FP
Van	0.001	96.41%	46	19
Van	0.00261	99.88%	48	13
Van	0.0001	95.41%	46	7
Van	0.00001	97.65%	45	14
Truck	0.001	69.84%	44	18
Truck	0.00261	71.13%	46	9
Truck	0.0001	88.93%	55	17
Truck	0.00001	51.87%	16	2
Car	0.001	99.57%	46	4
Car	0.00261	99.71%	47	22
Car	0.0001	100.00%	44	0
Car	0.00001	90.90%	42	6

1.4 Training Results: Dataset 2

Table 44: YOLOv4-Tiny Model Training Results: Dataset 2

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	54.46%	0.0536	68.14%	0.83	0.52	0.64	88	18	82
0.001	58.32%	0.0748	65.47%	0.80	0.51	0.62	86	22	84
0.0001	**73.37%**	**0.1960**	**62.29%**	**0.79**	**0.56**	**0.66**	**96**	**25**	**74**
0.00001	66.51%	0.6737	56.59%	0.83	0.48	0.61	82	17	88

Table 45: YOLOv4-Tiny Model Training Results by Class: Dataset 2

Class	Learning Rate	mAP	TP	FP
Van	0.001	37.31%	7	0
Van	0.00261	22.40%	8	0
Van	0.0001	42.97%	15	0
Van	0.00001	38.45%	4	2
Truck	0.001	47.75%	34	1
Truck	0.00261	48.10%	34	0
Truck	0.0001	90.21%	41	1
Truck	0.00001	69.82%	35	3
Car	0.001	89.90%	45	21
Car	0.00261	92.89%	46	18
Car	0.0001	86.93%	40	24
Car	0.00001	91.25%	43	12

Table 46: YOLOv4 Model-Tiny Training Results: Dataset 2 + DCGAN Images

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	73.39%	0.0645	66.49%	0.82	0.59	0.69	101	22	69
0.001	80.22%	0.0907	68.62%	0.86	63	0.73	107	18	63
0.0001	89.13%	0.2744	71.89%	0.95	0.72	0.82	122	7	48
0.00001	63.61%	0.7522	57.82%	0.81	0.49	0.61	83	19	87

Table 47: YOLOv4-Tiny Model Training Results by Class: Dataset 2 + DCGAN Images

Class	Learning Rate	mAP	TP	FP
Van	0.001	63.02%	23	12
Van	0.00261	52.45%	20	19
Van	0.0001	81.21%	30	3
Van	0.00001	31.22%	11	14
Truck	0.001	77.73%	37	0
Truck	0.00261	68.17%	34	1
Truck	0.0001	88.76%	46	0
Truck	0.00001	68.94%	33	1
Car	0.001	99.91%	47	6
Car	0.00261	99.56%	47	2
Car	0.0001	97.41%	46	4
Car	0.00001	90.67%	39	4

Table 48: YOLOv4-Tiny Model Training Results: Dataset 2 + cDCGAN Images

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	62.71%	0.0713	63.55%	0.77	0.54	0.64	92	27	78
0.001	64.99%	0.0907	68.19%	0.84	0.54	0.66	92	17	78
0.0001	74.17%	0.2744	61.92%	0.81	0.58	0.68	100	24	70
0.00001	66.67%	0.6895	63.12%	0.91	0.62	0.52	86	9	84

Table 49: YOLOv4-Tiny Model Training Results by Class: Dataset 2 + cDCGAN Images

Class	Learning Rate	mAP	TP	FP
Van	0.001	35.96%	12	6
Van	0.00261	26.90%	11	17
Van	0.0001	45.22%	14	5
Van	0.00001	46.30%	10	2
Truck	0.001	65.29%	36	1
Truck	0.00261	62.79%	34	1
Truck	0.0001	81.90%	40	0
Truck	0.00001	61.66%	33	0
Car	0.001	93.70%	44	10
Car	0.00261	98.44%	37	9
Car	0.0001	95.39%	46	19
Car	0.00001	92.06%	43	7

Table 50: YOLOv4-Tiny Model Training Results: Dataset 2 + WGAN Images

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	68.09%	0.0959	68.34%	0.83	0.55	0.66	94	19	76
0.001	68.51%	0.0636	74.81%	0.90	0.54	0.68	92	10	78
0.0001	73.40%	0.2675	64.09%	0.82	0.58	0.68	99	21	71
0.00001	63.85%	0.7461	63.60%	0.88	0.48	0.62	82	11	88

Table 51: YOLOv4-Tiny Model Training Results by Class: Dataset 2 + WGAN Images

Class	Learning Rate	mAP	TP	FP
Van	0.001	32.68%	13	12
Van	0.00261	38.40%	11	0
Van	0.0001	45.20%	12	3
Van	0.00001	36.63%	9	6
Truck	0.001	73.26%	34	0
Truck	0.00261	68.38%	34	0
Truck	0.0001	84.32%	44	0
Truck	0.00001	66.64%	32	0
Car	0.001	98.33%	47	7
Car	0.00261	98.77%	47	10
Car	0.0001	90.96%	43	18
Car	0.00001	88.47%	41	5

Table 52: YOLOv4-Tiny Model Training Results: Dataset 2 + cWGAN Images

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	59.42%	0.0705	77.55%	0.95	0.54	0.69	92	5	78
0.001	63.53%	0.1033	74.22%	0.91	0.54	0.68	92	9	78
0.0001	73.04%	0.2463	53.50%	0.68	0.54	0.60	92	44	78
0.00001	64.87%	0.6813	63.28%	0.90	0.49	0.64	84	9	86

Table 53: YOLOv4-Tiny Model Training Results by Class: Dataset 2 + cWGAN Images

Class	Learning Rate	mAP	TP	FP
Van	0.001	29.45%	11	2
Van	0.00261	23.35%	11	1
Van	0.0001	45.00%	12	0
Van	0.00001	31.02%	8	3
Truck	0.001	62.54%	34	0
Truck	0.00261	55.46%	34	0
Truck	0.0001	82.93%	34	1
Truck	0.00001	66.86%	32	2
Car	0.001	98.61%	47	7
Car	0.00261	99.46%	47	4
Car	0.0001	91.19%	46	43
Car	0.00001	96.72%	44	4

1.5 Training Results: Combined Dataset

Table 54: YOLOv4-Tiny Model Training Results: Combined Dataset

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	97.75%	0.0449	79.13%	0.94	0.97	0.95	165	11	5
0.001	**98.53%**	**0.0793**	**76.58%**	**0.94**	**0.98**	**0.96**	**167**	**11**	**3**
0.0001	98.51%	0.2032	74.07%	0.91	0.98	0.94	167	17	3
0.00001	93.37%	0.5752	64.85%	0.90	0.84	0.87	143	16	27

Table 55: YOLOv4-Tiny Model Training Results by Class: Combined Dataset

Class	Learning Rate	mAP	TP	FP
Van	0.001	97.47%	45	2
Van	0.00261	95.15%	43	0
Van	0.0001	97.31%	45	1
Van	0.00001	94.63%	44	5
Truck	0.001	99.89%	75	3
Truck	0.00261	99.92%	75	5
Truck	0.0001	99.93%	75	4
Truck	0.00001	91.00%	56	9
Car	0.001	98.24%	47	6
Car	0.00261	98.17%	47	6
Car	0.0001	98.29%	47	12
Car	0.00001	94.49%	43	2

Table 56: YOLOv4-Tiny Model Training Results: Combined Dataset + DCGAN Images

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	100.00%	0.0765	80.08%	0.96	1.00	0.98	170	0	7
0.001	100.00%	0.0963	79.82%	0.96	1.00	0.98	170	0	8
0.0001	98.68%	0.2314	77.09%	0.96	0.99	0.98	169	1	7
0.00001	91.89%	0.6048	63.80%	0.89	0.71	0.79	121	15	49

Table 57: YOLOv4-Tiny Model Training Results by Class: Combined Dataset + DCGAN Images

Class	**Learning Rate**	**mAP**	**TP**	**FP**
Van	0.001	100.00%	48	0
Van	0.00261	100.00%	48	6
Van	0.0001	95.88%	47	1
Van	0.00001	89.25%	47	10
Truck	0.001	100.00%	75	4
Truck	0.00261	100.00%	75	5
Truck	0.0001	100.00%	37	4
Truck	0.00001	92.18%	43	1
Car	0.001	100.00%	47	4
Car	0.00261	100.00%	47	2
Car	0.0001	91.91%	47	2
Car	0.00001	94.25%	41	4

Table 58: YOLOv4-Tiny Model Training Results: Combined Dataset + cDCGAN Images

Learning Rate	**mAP**	**CIoU Loss**	**Avg. IoU**	**Precision**	**Recall**	**F1-Score**	**TP**	**FP**	**FN**
0.00261	99.26%	0.0801	79.76%	0.95	0.99	0.97	169	9	1
0.001	99.26%	0.1006	81.27%	0.97	0.99	0.98	169	6	1
0.0001	98.98%	0.2587	78.25%	0.97	0.99	0.98	169	5	1
0.00001	93.32%	0.6844	67.76%	0.95	0.72	0.82	123	7	47

Table 59: YOLOv4-Tiny Model Training Results by Class: Combined Dataset + cDCGAN Images

Class	**Learning Rate**	**mAP**	**TP**	**FP**
Van	0.001	97.79%	47	1
Van	0.00261	97.92%	47	0
Van	0.0001	100.00%	48	0
Van	0.00001	90.26%	39	3
Truck	0.001	99.98%	75	4
Truck	0.00261	100.00%	75	5
Truck	0.0001	99.95%	74	3
Truck	0.00001	93.16%	41	2
Car	0.001	100.00%	47	1
Car	0.00261	99.87%	47	4
Car	0.0001	100.00%	47	2
Car	0.00001	96.55%	43	2

Table 60: YOLOv4-Tiny Model Training Results: Combined Dataset + WGAN Images

Learning Rate	**mAP**	**CIoU Loss**	**Avg. IoU**	**Precision**	**Recall**	**F1-Score**	**TP**	**FP**	**FN**
0.00261	100.00%	0.0678	81.56%	0.97	1.00	0.99	170	5	0
0.001	99.99%	0.0678	81.49%	0.97	1.00	0.99 170	5	0	
0.0001	98.79%	0.2626	78.19%	0.99	0.99	0.98	168	6	2
0.00001	93.32%	0.6224	66.00%	0.92	0.79	0.85	135	12	35

Table 61: YOLOv4-Tiny Model Training Results by Class: Combined Dataset + WGAN Images

Class	**Learning Rate**	**mAP**	**TP**	**FP**
Van	0.001	100.00%	48	0
Van	0.00261	100.00%	48	0
Van	0.0001	96.41%	46	2
Van	0.00001	95.80%	41	7
Truck	0.001	99.98%	75	4
Truck	0.00261	100.00%	75	4
Truck	0.0001	99.97%	75	3
Truck	0.00001	87.84%	50	4
Car	0.001	100.00%	47	1
Car	0.00261	100.00%	47	1
Car	0.0001	100.00%	47	1
Car	0.00001	99.24%	44	1

Table 62: YOLOv4-Tiny Model Training Results: Combined Dataset + cWGAN Images

Learning Rate	**mAP**	**CIoU Loss**	**Avg. IoU**	**Precision**	**Recall**	**F1-Score**	**TP**	**FP**	**FN**
0.00261	100.00%	0.0595	80.39%	0.96	1.00	0.99	170	8	0
0.001	99.99%	0.0791	80.95%	0.97	1.00	0.99	170	5	0
0.0001	97.73%	0.2544	76.29%	0.94	0.98	0.96	167	10	3
0.00001	91.45%	0.6499	64.56%	0.89	0.75	0.82	128	16	42

Table 63: YOLOv4-Tiny Model Training Results by Class: Combined Dataset + cWGAN Images

Class	Learning Rate	mAP	TP	FP
Van	0.001	100.00%	48	0
Van	0.00261	100.00%	48	0
Van	0.0001	96.24%	46	2
Van	0.00001	94.50%	41	6
Truck	0.001	99.97%	75	4
Truck	0.00261	100.00%	75	5
Truck	0.0001	99.91%	75	5
Truck	0.00001	81.10%	43	8
Car	0.001	100.00%	47	1
Car	0.00261	100.00%	47	3
Car	0.0001	97.04%	46	3
Car	0.00001	97.95%	44	2

1.6 Experiment I Results

Table 64: Experiment I Results: Learning Rate = 0.001

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	70.65%	0.0843	51.46%	0.70	0.64	0.67	183	78	103
Combined + DCGAN	77.59%	0.1035	58.56%	0.78	0.60	0.60	171	47	115
Combined + cDCAN	61.16%	0.1001	38.39%	0.53	0.52	0.53	150	133	136
Combined + WGAN	62.30%	0.0954	46.09%	0.62	0.51	0.56	145	88	141
Combined + cWGAN	68.40%	0.0930	50.22%	0.68	0.58	0.63	166	79	120

Table 65: YOLOv4-Tiny Model Training Results by Class: Experiment I

Dataset	Class	mAP	TP	FP
Combined	Van	74.22%	37	11
Combined + DCGAN	Van	83.65%	65	22
Combined + cDCGAN	Van	69.82%	25	1
Combined + WGAN	Van	68.87%	25	3
Combined + cWGAN	Van	87.65%	52	3
Combined	Truck	75.35%	68	10
Combined + DCGAN	Truck	74.87%	55	7
Combined + cDCGAN	Truck	73.55%	58	21
Combined + WGAN	Truck	74.47%	61	10
Combined + cWGAN	Truck	67.84%	60	19
Combined	Car	62.39%	78	57
Combined + DCGAN	Car	74.23%	51	18
Combined + cDCGAN	Car	40.12%	67	111
Combined + WGAN	Car	43.57%	59	75
Combined + cWGAN	Car	49.72%	54	57

1.7 Experiment 1I Results

Table 66: Experiment II Results: Learning Rate = 0.00261 | Iterations = 6,000

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	61.59%	0.0587	43.31%	0.59	0.64	0.54	154	107	132
Combined + cDCAN	57.73%	0.0876	34.08%	0.45	0.36	0.53	125	182	182
Combined + WGAN	56.68%	0.0798	40.11%	0.54	0.49	0.56	140	120	146
Combined + cWGAN	53.60%	0.0798	37.24%	0.49	0.40	0.63	113	118	173

Table 67: Experiment II Results by Class: Learning Rate = 0.00261 | Iterations = 6,000

Dataset	Class	mAP	TP	FP
Combined	Van	57.74%	23	12
Combined + cDCGAN	Van	75.28%	34	0
Combined + WGAN	Van	60.71%	18	1
Combined + cWGAN	Van	58.46%	17	1
Combined	Truck	86.30%	77	8
Combined + cDCGAN	Truck	67.08%	26	6
Combined + WGAN	Truck	74.86%	67	15
Combined + cWGAN	Truck	66.14%	38	10
Combined	Car	40.17%	54	87
Combined + cDCGAN	Car	30.82%	44	119
Combined + WGAN	Car	34.46%	55	104
Combined + cWGAN	Car	36.19%	58	107

Table 68: Experiment II Results: Learning Rate = 0.00261 | Iterations = 2,000

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	60.72%	0.1218	42.04%	0.70	0.58	0.54	142	101	144
Combined + cDCAN	74.97%	0.2200	47.54%	0.66	0.50	0.57	142	74	144
Combined + WGAN	63.64%	0.1903	38.03%	0.53	0.36	0.43	103	92	183
Combined + cWGAN	78.49%	0.1820	57.37%	0.79	0.59	0.67	168	44	118

Table 69: Experiment II Results: Learning Rate = 0.00261 | Iterations = 2,000

Dataset	Class	mAP	TP	FP
Combined	Van	53.42%	6	10
Combined + cDCGAN	Van	85.97%	26	0
Combined + WGAN	Van	80.12%	14	0
Combined + cWGAN	Van	89.45%	50	0
Combined	Truck	77.70%	68	20
Combined + cDCGAN	Truck	64.18%	39	20
Combined + WGAN	Truck	69.40%	38	17
Combined + cWGAN	Truck	66.47%	58	20
Combined	Car	51.04%	68	71
Combined + cDCGAN	Car	74.75%	77	54
Combined + WGAN	Car	41.39%	51	75
Combined + cWGAN	Car	79.56%	60	24

Table 70: Experiment II Results: Learning Rate = 0.00261 | Iterations = 4,000

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	70.10%	0.07108	47.65%	0.66	0.68	0.67	195	101	91
Combined + cDCAN	53.57%	01079	35.06%	0.48	0.41	0.45	118	126	168
Combined + WGAN	67.58%	00520	45.67%	0.57	0.62	0.57	150	92	136
Combined + cWGAN	58.29%	00874	38.87%	0.52	0.46	0.49	131	120	155

Table 71: Experiment II Results: Learning Rate = 0.00261 | Iterations = 4,000

Dataset	Class	mAP	TP	FP
Combined	Van	64.82%	43	21
Combined + cDCGAN	Van	46.08%	5	0
Combined + WGAN	Van	69.52%	22	4
Combined + cWGAN	Van	61.42%	27	3
Combined	Truck	76.13%	76	28
Combined + cDCGAN	Truck	66.89%	36	18
Combined + WGAN	Truck	75.84%	56	11
Combined + cWGAN	Truck	73.60%	33	13
Combined	Car	69.36%	76	52
Combined + cDCGAN	Car	47.73%	77	108
Combined + WGAN	Car	57.39%	72	77
Combined + cWGAN	Car	39.83%	71	104

Table 72: Experiment II Results: Learning Rate = 0.00522 | Iterations = 2,000

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	67.19%	0.1138	47.95%	0.67	0.54	0.60	155	78	131
Combined + cDCAN	70.06%	0.1915	43.78%	0.51	0.51	0.56	147	96	139
Combined + WGAN	68.06%	0.1659	45.08%	0.49	0.55	0.57	139	81	147
Combined + cWGAN	67.72%	0.1544	42.41%	0.52	0.55	0.49	148	106	138

Table 73: Experiment Results by Class: Learning Rate = 0.00522 | Iterations = 2,000

Dataset	Class	mAP	TP	FP
Combined	Van	64.07%	23	6
Combined + cDCGAN	Van	79.87%	15	1
Combined + WGAN	Van	79.57%	23	4
Combined + cWGAN	Van	83.98%	37	4
Combined	Truck	81.86%	73	14
Combined + cDCGAN	Truck	68.72%	70	26
Combined + WGAN	Truck	75.28%	65	17
Combined + cWGAN	Truck	75.83%	58	27
Combined	Car	55.80%	59	58
Combined + cDCGAN	Car	61.59%	62	69
Combined + WGAN	Car	49.34%	51	60
Combined + cWGAN	Car	43.36%	53	75

Table 74: Experiment II Results: Learning Rate = 0.00522 | Iterations = 4,000

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	74.71%	0.0618	56.69%	0.74	0.69	0.71	196	68	90
Combined + cDCAN	60.32%	0.1116	39.66%	0.54	0.50	0.52	142	119	144
Combined + WGAN	57.35%	0.0763	38.52%	0.52	0.47	0.49	133	121	153
Combined + cWGAN	62.52%	0.0782	41.15%	0.56	0.47	0.51	135	108	151

Table 75: Experiment II Results:Learning Rate = 0.00522 | Iterations = 4,000

Dataset	Class	mAP	TP	FP
Combined	Van	82.53%	55	17
Combined + cDCGAN	Van	67.31%	16	0
Combined + WGAN	Van	50.03%	19	16
Combined + cWGAN	Van	69.59%	22	1
Combined	Truck	84.35%	87	8
Combined + cDCGAN	Truck	74.15%	71	26
Combined + WGAN	Truck	84.88%	56	15
Combined + cWGAN	Truck	73.56%	45	28
Combined	Car	57.25%	54	43
Combined + cDCGAN	Car	39.50%	55	93
Combined + WGAN	Car	37.13%	58	90
Combined + cWGAN	Car	44.41%	68	87

Table 76: Experiment II Results: Learning Rate = 0.001305 | Iterations = 2,000

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	70.12%	0.1116	46.28%	0.66	0.52	0.59	150	76	136
Combined + cDCAN	74.79%	0.2342	53.91%	0.76	0.46	0.57	131	41	155
Combined + WGAN	70.45%	0.2333	52.54%	0.73	0.50	0.59	143	52	143
Combined + cWGAN	75.68%	0.1990	51.36%	0.73	0.57	0.64	162	60	124

Table 77: Experiment II Results: Learning Rate = 0.001305 | Iterations = 2,000

Dataset	Class	mAP	TP	FP
Combined	Van	65.24%	8	2
Combined + cDCGAN	Van	81.06%	18	0
Combined + WGAN	Van	71.38%	30	2
Combined + cWGAN	Van	82.41%	27	0
Combined	Truck	78.42%	76	21
Combined + cDCGAN	Truck	65.07%	43	10
Combined + WGAN	Truck	56.64%	43	26
Combined + cWGAN	Truck	71.14%	63	16
Combined	Car	66.70%	66	53
Combined + cDCGAN	Car	78.25%	70	31
Combined + WGAN	Car	83.33%	70	24
Combined + cWGAN	Car	73.49%	72	44

Table 78: Experiment II Results: Learning Rate = 0.001305 | Iterations = 4,000

Dataset	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
Combined	74.94%	0.0847	52.38%	0.73	0.69	0.71	197	73	89
Combined + cDCAN	76.52%	0.1312	53.92%	0.73	0.64	0.68	183	68	103
Combined + WGAN	59.42%	0.1324	37.37%	0.50	0.45	0.47	128	126	158
Combined + cWGAN	65.62%	0.1114	42.52%	0.57	0.54	0.55	154	117	132

Table 79: Experiment II Results: Learning Rate = 0.001305 | Iterations = 4,000

Dataset	Class	mAP	TP	FP
Combined	Van	76.18%	46	12
Combined + cDCGAN	Van	82.27%	37	0
Combined + WGAN	Van	69.53%	22	4
Combined + cWGAN	Van	74.62%	24	0
Combined	Truck	78.42%	80	11
Combined + cDCGAN	Truck	74.47%	69	11
Combined + WGAN	Truck	70.09%	50	16
Combined + cWGAN	Truck	80.67%	58	16
Combined	Car	60.81%	71	50
Combined + cDCGAN	Car	72.82%	77	57
Combined + WGAN	Car	38.64%	56	106
Combined + cWGAN	Car	41.56%	72	101

1.8 Experiment III Results

Table 80: Experiment III Results: Blended Augmentation | Alien Test Set

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	100.00	0.1060	80.57%	0.96	1.00	0.98	170	8	0
0.001	100.00%	0.1342	82.36%	0.97	1.00	0.99	170	5	0
0.0001	100.00%	0.3618	79.59%	0.98	1.00	0.99	170	4	0
0.00001	95.94%	0.8846	69.38%	0.95	0.69	0.80	118	6	52

Table 81: Experiment III Results by Class: Blended Augmentation | Original Test Set

Class	Learning Rate	mAP	TP	FP
Van	0.00261	100%	48	0
Van	0.001	100.00%	48	0
Van	0.0001	100.00%	48	0
Van	0.00001	96.63%	39	2
Truck	0.00261	100.00%	75	6
Truck	0.001	100.00%	75	4
Truck	0.0001	100.00%	75	2
Truck	0.00001	92.47%	39	1
Car	0.00261	100.00%	47	2
Car	0.001	100.00%	47	1
Car	0.0001	100.00%	47	2
Car	0.00001	98.70%	40	3

Table 82: Experiment III Results: Blended Augmentation | Alien Test Set

Learning Rate	mAP	CIoU Loss	Avg. IoU	Precision	Recall	F1-Score	TP	FP	FN
0.00261	71.28	0.1057	53.64%	0.76	0.61	0.68	175	55	111
0.001	85.67%	0.1464	60.24%	0.84	0.71	0.77	203	40	83
0.0001	80.37%	0.3671	58.82%	0.81	0.69	0.74	197	47	89
0.00001	57.53%	0.8848	52.08%	0.79	0.39	0.52	111	30	175

Table 83: Experiment III Results by Class: Blended Augmentation | Alien Test Set

Class	Learning Rate	mAP	TP	FP
Van	0.00261	80.46%	38	1
Van	0.001	90.09%	53	2
Van	0.0001	91.72%	45	1
Van	0.00001	66.87%	27	3
Truck	0.00261	76.80%	80	11
Truck	0.001	75.53%	77	21
Truck	0.0001	67.49%	74	21
Truck	0.00001	33.51%	16	9
Car	0.00261	56.59%	57	43
Car	0.001	91.41%	73	17
Car	0.0001	81.90%	78	25
Car	0.00001	72.20%	68	18

Bibliography

1. Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2016.

2. Alberto Cano. A survey on graphic processing unit computing for large-scale data mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(1):e1232, 2018.

3. Young W. Frimel S. Bihl, T. J. Artificial neural networks and data science. *Encyclopedia of Data Science and Machine Learning, IGI Global*, To Appear.

4. Aurelien Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Sebastopol, CA, 2017.

5. Seongdeok Bang, Francis Baek, Somin Park, Wontae Kim, and Hyoungkwan Kim. Image augmentation to improve construction resource detection using generative adversarial networks, cut-and-paste, and image transformation techniques. *Automation in Construction*, 115:103198, 2020.

6. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

7. Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

8. Francesco Tacchino, Chiara Macchiavello, Dario Gerace, and Daniele Bajoni. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1):1–8, 2019.

9. Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

10. David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

11. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

12. Anthony Baietto, Jayson Boubin, Patrick Farr, Trevor J. Bihl, Aaron M. Jones, and Christopher Stewart. Lean neural networks for autonomous radar waveform design. *Sensors*, 22(4), 2022.

13. Werbos. Backpropagation: past and future. In *IEEE 1988 International Conference on Neural Networks*, pages 343–353 vol.1, 1988.

14. P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

15. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

16. Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: comparison of trends in practice and research for deep learning. In *2nd International Conference on Computational Sciences and Technology*, pages 124–133, 2021.

17. Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

18. Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

19. B Willmore and DJ Tolhurst. Characterizing the sparseness of neural codes. *Network (Bristol, England)*, 12(3):255—270, August 2001.

20. Diganta Misra. Mish: A self regularized non-monotonic activation function. In *BMVC*, 2020.

21. Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

22. Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

23. Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

24. Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

25. Fei Wang, Zhanyao Zhang, Chun Liu, Yili Yu, Songling Pang, Neven Duić, Miadreza Shafie-Khah, and João PS Catalão. Generative adversarial networks and convolutional neural networks based weather classification model for day ahead short-term photovoltaic power forecasting. *Energy conversion and management*, 181:443–462, 2019.

26. Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):826–834, 1983.

27. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

28. David E Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34, 1995.

29. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

30. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

31. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

32. Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceed-*

ings of the IEEE conference on computer vision and pattern recognition, pages 580–587, 2014.

33. Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

34. Asifullah Khan, Anabia Sohail, Umme Zahoora, and A. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, pages 1 – 62, 2020.

35. Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

36. Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 111–118, Madison, WI, USA, 2010. Omnipress.

37. Aurelien Geron. *Hands On Machine Learning With Scikit Learn & TensorFlow*. O Reilly Media Inc., Sebastopol, 2017.

38. Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

39. Li Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 Conference on Computer Vision and Pattern Recognition Workshop*, pages 178–178, 2004.

40. Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'04, page 97–104, USA, 2004. IEEE Computer Society.

41. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

42. Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

43. Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

44. Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection. *Advances in Neural Information Processing Systems*, 32:6642–6652, 2019.

45. Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.

46. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

47. Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Confer-*

ence on Computer Vision and Pattern Recognition (CVPR), pages 5987–5995, 2017.

48. Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

49. Robert J. Wang, Xiang Li, Shuang Ao, and Charles X. Ling. Pelee: A real-time object detection system on mobile devices. In *NeurIPS*, 2018.

50. Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.

51. Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. *Advances in Neural Information Processing Systems*, 31:10727–10737, 2018.

52. Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? *Advances in Neural Information Processing Systems*, 32:4694–4703, 2019.

53. Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *Learning*, 10:3.

54. Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

55. Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13029–13038, 2021.

56. Han Qiu, Yuchen Ma, Zeming Li, Songtao Liu, and Jian Sun. Borderdet: Border feature for dense object detection. In *European Conference on Computer Vision*, pages 549–564. Springer, 2020.

57. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

58. Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

59. Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

60. Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

61. Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

62. Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous

methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

63. Clay Sheppard and Maryam Rahnemoonfar. Real-time scene understanding for uav imagery based on deep convolutional neural networks. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 2243–2246, 2017.

64. Hojjat Salehinejad, Shahrokh Valaee, Tim Dowdell, Errol Colak, and Joseph Barfett. Generalization of deep neural networks for chest pathology classification in x-rays using generative adversarial networks. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 990–994. IEEE, 2018.

65. Jiaquan Shen, Ningzhong Liu, Han Sun, and Huiyu Zhou. Vehicle detection in aerial images based on lightweight deep convolutional network and generative adversarial network. *IEEE Access*, 7:148119–148130, 2019.

66. Victor-Emil Neagoe and Paul Diaconescu. Cnn hyperspectral image classification using training sample augmentation with generative adversarial networks. In *2020 13th International Conference on Communications (COMM)*, pages 515–519, 2020.

67. Tzutalin. Labelimg. Free Software: MIT License, 2015.

68. Rafael Padilla, Wesley L Passos, Thadeu LB Dias, Sergio L Netto, and Eduardo AB da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3):279, 2021.

69. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

70. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

71. Jake Cowton, Ilias Kyriazakis, and Jaume Bacardit. Automated individual pig localisation, tracking and behaviour metric extraction using deep learning. *IEEE Access*, PP, 08 2019.

72. Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

73. Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12993–13000, 2020.

74. Aladdin Persson. Machine-learning-collection/ml/pytorch/gans/2. dcgan at master· aladdinpersson/machine-learning-collection · github. `https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/GANs/2.%20DCGAN`.

75. The GIMP Development Team. Gimp.

76. Jason Brownlee. *Generative adversarial networks with python: deep learning generative models for image synthesis and image translation*. Machine Learning Mastery, 2019.

Acronyms

Adam Adaptive Moment Estimation. 14, 41, 43, 66

ANN Artificial Neural Network. 9, 10, 11, 19, 25

ANNs Artifical Neural Networks. 2, 8, 12, 15

AP Average Precision. 33, 34, 63

BoF Bag of Freebies. 28, 31

BoS Bag of Specials. 29, 32

cDCGAN Conditional Deep Convolutional Generative Adversarial Network. 65, 69, 70, 75, 76, 78, 81, 82, 83, 84, 89, 92, 94, 112, 123

cGAN Conditional Generative Adversarial Network. 38, 49

cGANs Conditional Generative Adversarial Networks. 38

CIoU Complete Intersection over Union. xiii, xiv, 31, 61, 64, 65, 97, 98, 99, 101, 102, 103, 104, 105, 106, 107, 108, 109, 123, 124, 125, 126, 127, 128, 131, 132, 134, 136

CmBN Cross mini-Batch Normalization. 31, 32

CNN Convolutional Neural Network. viii, 5, 6, 7, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 37, 39, 44, 45, 46, 49, 135, 138

CNNs Convolutional Neural Networks. 2, 6, 8, 19, 20, 21, 22, 28, 35, 44, 45, 46, 134

CPU Central Processing Unit. 6

CSPNet Cross Stage Partial Network. 26, 27, 28, 29

cWGAN Conditional Wasserstein Generative Adversarial Network. 65, 73, 80, 81, 84, 85, 86, 90, 91, 93, 95, 97, 99, 100, 105, 108, 112, 123, 128

DCGAN Deep Convolutional Generative Adversarial Network. 35, 39, 41, 43, 44, 65, 66, 67, 68, 69, 70, 71, 72, 75, 76, 78, 80, 81, 83, 84, 85, 88, 89, 90, 91, 92, 93, 96, 101, 103, 119, 122, 123

DCNN Deep Convolutional Neural Network. 50

DIoU Distance Intersection over Union. 32

DNN Deep Neural Network. 11, 17

EM Earth Mover. 36, 41, 43, 44

FN False Negative. v, 62, 98, 100, 102, 104, 106, 108, 119, 124, 128, 129, 1

FP False Positive. v, 62, 98, 100, 102, 104, 106, 108, 109, 120, 124, 125, 128, 129, 130, 1

FPN Feature Pyramid Network. 30

FPS Frames Per Second. 33, 34

GAN Generative Adversarial Network. iv, v, 3, 4, 5, 6, 34, 35, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 52, 61, 65, 66, 67, 73, 74, 75, 76, 77, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 93, 95, 96, 99, 100, 108, 109, 110, 112, 117, 118, 119, 120, 121, 123, 124, 126, 127, 129, 133, 134, 135, 136, 137, 138, 139, 1

GANs Generative Adversarial Networks. 7, 8, 36, 38, 40, 41, 52, 65, 74, 75, 76, 77, 78, 79, 80, 81, 83, 84, 85, 88, 96, 97, 101, 105, 107, 108, 112, 114, 116, 118, 122, 123, 124, 126, 127, 132, 133, 134, 135, 136, 137, 138

GIMP GNU Image Manipulation Program. 73, 74

GPU Graphics Processing Unit. 2, 6, 20, 33, 66, 75

GRU Gated Recurrent Units. 50

IoU Intersection over Union. v, 5, 31, 32, 62, 63, 64, 97, 100, 101, 102, 104, 105, 106, 108, 109, 119, 124, 127, 128, 134, 135, 1

JS Jensen-Shannon. 37, 41

KL Kullback-Leibler. 36, 37, 41

LReLU Leaky Rectified Linear Unit. 16, 17, 19, 39, 43, 50, 69, 72, 137, 138

LSUN Large-Scale Scene Understanding. 39, 43

LTU Linear Threshold Unit. 9, 10

LTUs Linear Threshold Units. 11

mAP Mean Average Precision. iv, xiii, xiv, 49, 61, 63, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 119, 120, 121, 122, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 1

mAP@.50 mean average precision at IOU threshold .5. 18, 19, 63

MC-GAN Multi-conditioned Constrained Generative Adversarial Network. 49

MiWRC Multi-input weight residual connections. 29

ML machine learning. iv, 1, 2, 3, 4, 5, 6, 8, 12, 21, 33, 46, 134, 136, 1

MLP Multilayer Perceptron. 10, 43, 46

MS-COCO Microsoft Common Objects in Context. 18, 34

NMS non-maximum supression. 32

OA Overall Accuracy. 46

PA Product Accuracy. 46

PANet Path Aggregation Network. 26, 29, 30, 32

PGGAN Progressive Growing Generative Adversarial Network. 137

R-CNN Regions with Convolutional Neural Networks. 21, 47, 48

RAM Random-access Memory. 2, 6, 66, 75

ReLU Rectified Linear Unit. 15, 16, 17, 18, 19, 20, 40, 43, 44, 45, 50, 68, 137, 138

RGB Red, Green, Blue. 22, 67, 68, 78, 80, 81, 84, 85, 88, 96

RMSProp Root Mean Squared Propagation. 13, 14, 41, 42, 43

SAM Spatial Attention Module. 32

SAT Self-Adversarial Training. 31, 32

SPP Spatial Pyramid Pooling. 26, 29, 32

SVM Support Vector Machines. 21, 45

tanh Hyperbolic Tangent. 17, 18, 24, 40, 68

TP True Positive. v, 62, 98, 100, 102, 104, 106, 108, 119, 120, 124, 125, 128, 129, 130, 1

UA User Accuracy. 46

UAV Unmanned Aerial Vehicle. iv, 3, 7, 44, 50, 115, 137, 1

UAVs Unmanned Aerial Vehicles. 130, 133

WGAN Wasserstein Generative Adversarial Network. 36, 41, 42, 43, 44, 65, 67, 71, 72, 73, 76, 77, 78, 85, 90, 91, 92, 94, 95, 96, 112

WGAN-GP Wasserstein Generative Adversarial Network with Gradient Penalty. 46

YOLOv3 You Only Look Once Version 3. 25, 26, 27, 28, 31, 33

YOLOv4 You Only Look Once Version 4. 19, 25, 26, 27, 29, 31, 33, 61

YOLOv4-Tiny You Only Look Once Version 4 - Tiny. iv, 8, 34, 52, 60, 61, 62, 63, 64, 74, 97, 110, 112, 116, 117, 118, 126, 127, 129, 135, 136, 1

Glossary

Alien Test Set The test set containing the images not relating to the original training sets. This test set contains no images used for training the Generative Adversarial Networks. 111, 112, 116, 118, 119, 121, 122, 123, 124, 127, 128, 130, 132, 133, 134, 135, 136

Blended Augmentation Training Set The training set composed of of the original training set images and images augmented with images generated from every type of GAN. 117, 128, 130, 131, 132, 133, 135

child object An object located within a child image. 65, 73

child image An image contained inside of a parent dataset. 52, 54, 73, 74, 79

Combined Parent Dataset The dataset comprised of datasets 1 and 2. 112

combined dataset The combination of Dataset 1 and Dataset 2. 75, 105, 109, 118

Dataset 2 The dataset containing images of vehicles driving along a road captured at a sideways angle. 53, 56, 57, 75, 79, 81, 83, 91, 101, 109

Dataset 1 The dataset containing images of vehicles driving along a road captured at an overhead angle. 52, 53, 56, 75, 76, 79, 81, 83, 89, 90, 96, 97, 102, 109

Original Test Set The original test set containing the images related to the training set. This test set contains images used for training the Generative Adversarial Networks as well as images related to the training set. 112, 133

Parent Training Set The original training set. 74

parent image An image in a dataset containing a collection of the object's of interest. 65, 79, 114, 115

parent dataset An original dataset containing a collection of frames captured by the UAV . 52, 65, 66, 73, 74

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2022	Master's Thesis	Sept 2020 — Mar 2022

4. TITLE AND SUBTITLE

Using Generative Adversarial Networks to Augment Unmanned Aerial Vehicle Image Classification Training Sets

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

McCloskey, Benjamin, 2nd Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-22-M-151

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Trevor Bihl, DAF, DR-III, PhD
Sensors Directorate
Air Force Research Laboratory
2242 Avionics Circle
Wright-Patterson AFB, OH 45431
Email: trevor.bihl.2@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RYAR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

A challenging task in computer vision is finding techniques to improve the object detection and classification capabilities of ML models used for processing images acquired by moving aerial platforms. This research explores if GAN augmented UAV training sets can increase the generalizability of a detection model trained on said data. To answer this question, the YOLOv4-Tiny Object Detection Model was trained with aerial image training sets depicting rural environments. The salient objects within the frames were recreated using various GAN architectures, placed back into the original frames, and the augmented frames appended to the original training sets. GAN augmentation on aerial image training sets led to a 6.75% increase on average in the mAP of the YOLOv4-Tiny Object Detection model with a best-case increase of 15.76%. Similarly, a 4.13% increase on average and a best-case increase of 9.60% was observed for the IoU rate. Finally, 100.00% TP, 4.70% FP, and zero FN detection rates were yielded, providing further evidence supporting GAN augmentation for object detection model training sets. .

15. SUBJECT TERMS

artificial neural network (ANN), convolutional neural network (CNN), deep learning, computer vision

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Bruce Cox, AFIT/ENS
U	U	U	UU	208	19b. TELEPHONE NUMBER *(include area code)* (937) 785-3636 x4676; bruce.cox@afit.edu