Synthesizing Realistic Data for Vision Based Drone-to-Drone Detection

Sudha Ravali Yellapantula

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Ryan K. Williams, Chair

Pratap Tokekar

A. Lynn Abbott

June 17, 2019

Blacksburg, Virginia

Keywords: GANs, Deep Learning, Object Detection

Synthesizing Realistic Data for Vision Based Drone-to-Drone Detection

Sudha Ravali Yellapantula

(ABSTRACT)

In the thesis, we aimed at building a robust UAV(drone) detection algorithm through which, one drone could detect another drone in flight. Though this was a straight forward object detection problem, the biggest challenge we faced for drone detection is the limited amount of drone images for training. To address this issue, we used Generative Adversarial Networks, CycleGAN to be precise, for the generation of realistic looking fake images which were indistinguishable from real data. CycleGAN is a classic example of Image to Image Translation technique, and we applied this in our situation where synthetic images from one domain were transformed into another domain, containing real data. The model, once trained, was capable of generating realistic looking images from synthetic data without the presence of real images. Following this, we employed a state of the art object detection model, YOLO(You Only Look Once), to build a Drone Detection model that was trained on the generated images. Finally, the performance of this model was compared against different datasets in order to evaluate its performance.

Synthesizing Realistic Data for Vision Based Drone-to-Drone Detection

Sudha Ravali Yellapantula

(GENERAL AUDIENCE ABSTRACT)

In the recent years, technologies like Deep Learning and Machine Learning have seen many rapid developments. Among the many applications they have, object detection is one of the widely used application and well established problems. In our thesis, we deal with a scenario where we have a swarm of drones and our aim is for one drone to recognize another drone in it's field of vision. As there was no drone image dataset readily available, we explored different ways of generating realistic data to address this issue. Finally, we proposed a solution to generate realistic images using Deep Learning techniques and trained an object detection model on it where we evaluated how well it has performed against other models.

Dedication

To Mom and Dad. This is for you and I love you the most.

Acknowledgments

Firstly, I would like to express my sincere gratitude towards my advisor, Dr. Ryan Williams for his constant support and his belief in me. I would like to use this opportunity to thank him for his guidance, unmatched encouragement and the patience he has shown throughout this time. His passion towards his work has always inspired me and it will continue to do so. He taught me that it was important to love the work you do and I could not have asked for a better advisor.

I would like to thank Dr. Pratap Tokekar and Dr. Lynn Abbott for being on my advising committee and providing their support and feedback. I am grateful to Dr. Tokekar for being my interim advisor and guiding me in the beginning of my graduate education.

I would like to thank my lab members - Larkin, Pratik, Gavin and Jun for their help and support in my work. I would like to give a special acknowledgment to Daniel Monzel, for his invaluable help and expertise.

I would like to thank my friends, Pranavi, Yamini, Naina, Prashant, Vikram and Sourabh for being there for me and sharing this amazing journey for the past two years. Vignendra Jannela, thank you for your support, inspiration and humor from the time I've known you.

Most importantly, I would like to thank my parents for all the love, sacrifice and understanding they have shown since I can remember. I would also like to thank my Brother, Sister-in-law and Milo for making sure that I could count on them at anytime.

Contents

List of Figures

Chapter 1

Introduction to the Problem

1.1 Motivation

In the recent years, there have been rapid technological advancements in the fields of Deep Learning and Computer Vision. Among these, object detection is one such application that is pervasive in almost every practical application that involves vision. We deal with one such application where we have multiple agents who are tying to interact and coordinate with one another. To be precise, in our scenario, we have a swarm of drones where one drone is trying to interact with another drone in it's field of vision. In order to accomplish this, the first step in the process is to locate where a drone is in the field of vision of another drone. The solution to this problem seemed to be an object detection algorithm, to be more precise, a drone detection algorithm in the initial glance.

The current model we have takes the help of April tags[18] to approach this problem. A unique April tag was assigned to every drone in the network. This process used traditional Machine Learning techniques as opposed to the more advanced Deep Learning techniques. The model gave good results in under laboratory settings where there was uniform light distribution, no sudden movements and the clarity and position of the camera maintained. This was a good sign, but this approach failed to match up to these standards in the real world environment. The main reasons behind this digression of behaviour were observed to be unequal distribution of light on and around the drone surfaces, blurriness of the images while trying to capture the scene, inability of the April tags to be stationary without any movement due to their flimsy structure, etc.

These were the problems faced by the current model and this helped us realize that using traditional methods alone to solve this problem was not enough. We realized that to detect a new object like drone, it was necessary to rely on sources like the Deep Learning methods. It was expected that an object like a UAV that resembles our drone, is a hard object class to find in the pre-trained object detection datasets like COCO etc. This meant that there is an additional task of Data Collection to be handled before tackling the problem of Object Detection. There was a paper earlier on Drone Monitoring[19], but we realized that they collected a lot of data through videos to train the drone detection systems but end result and the type of data required was different in both the cases as they have used base station to collect the drone videos. Since there is no ready to use Drone dataset out there, it meant that we have to generate our own data for the later stages in the pipeline.

In order to minimize the human involvement and to achieve the baseline results as to what to expect, the object detection model was first tried out on a Synthetic dataset. We used one of the state of the art detection algorithms like YOLO(You Only Look Once)[20] for the detection purpose. The Synthetic dataset consisted of 3D rendered images of our Drone model (DJI Matrice 100) in the background that resembled the place we usually conduct our experiments with drones. The accuracy of the results were about 6 percent, which was not a good enough value that we expect our final model to have. This meant that realistic images were required for the purpose of building a custom drone detection model. One way to collect real images include flying two drones in order to obtain videos of the drone in flight, followed by converting it into images which should be cropped close enough to see the drone clearly. All of the above steps take a lot of time and resources, and obtaining a huge number of images from the above method is not an easy solution.

Generation of a large number of images is not a difficult task for Generative Adversarial Networks[21]. Recent advancements in this field gave rise to GANs which could produce images using Image to Image translation techniques. This was a better alternative to choose from in order to generate any number of different realistic looking images while greatly reducing the human effort. CycleGANs[8] were chosen for this purpose of generation of realistic data. The CycleGAN uses both Real and Synthetic datasets for the model to learn the distribution of each and convert images of one domain to another. Once the model is trained, we would only need synthetic images, images which are rendered through a 3D model, to obtain the realistic looking images that we use as the dataset for the Object Detection purpose. This is the proposed completely Deep Learning based solution to the problem, and through our experiments, we aimed to see if the real data could be supplemented by fake data.

1.2 Thesis Outline

The thesis is organized into five chapters.

In the chapter, ”Preliminaries and Related Work”, the basic introduction to the concepts that were made use of in this process is discussed along with the previous work that has been done in the same domain.

In the chapter, ”Solution Pipeline”, the proposed solution is explained which begins with the explanation as to why CycleGAN was chosen and how it works. Following this, we discussed how we integrated the results obtained from this process with the YOLO Object Detection Model. The working of YOLO is described in detail along with the metrics chosen to evaluate results from different datasets.

In the final chapter, ”Experiments and Results”, all the experiments conducted are discussed in detail. The results are tabulated and the observations made are listed.

The thesis is concluded with the conclusions drawn and observations made in the experiments and including the scope for future work.

Chapter 2

Preliminaries and Related Work

2.1 Neural Networks

Today, Deep Learning[22] is becoming responsible for producing one of the most advanced artificially intelligent systems we have in our world. Deep Learning is a specialized branch in Machine Learning and is surfacing to be one of the most sought out fields in Computer and Cognitive Science. The neural networks[23] are the building blocks of Deep Learning. These are essentially a huge collection of interconnected nodes with computing abilities which are densely connected. The nodes are stacked up into layers and together, they help in the formation of directed graph which is a neural network. The working of the neural network can be explained with an example, where a node in a certain later receives and computes data from several interconnected nodes in the previous layer, and sends the computed information to the nodes in the above layer.

By now, it can be understood how the neurons are the fundamental units of the neural network. To be precise, in Machine Learning, the node receives a certain number of values and bias as input. These input values are multiplied by the weights associated with them and is given to the output node along with the bias. This can be understood in a better way with the help of the diagram 2.1.

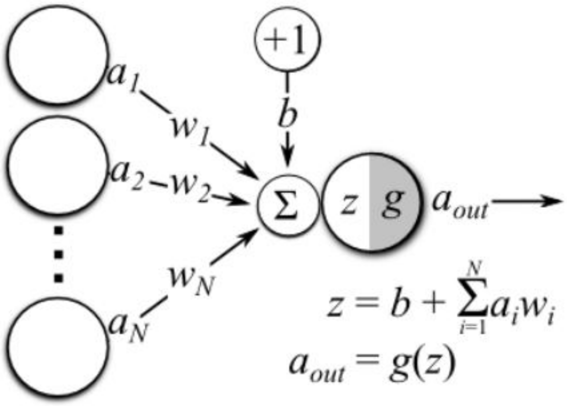


Figure 2.1: Operations in a neural network[1]

In the figure 2.1, it can be seen how each connection holds a certain weight and this get multiplied to the signal values in the following way. In the diagram, it can be seen that the output of the node is written as $g(z)$, which is known as the activation function. Since it is an output layer, let us assume this activation function to be sigmoid function which can be written as :

$$\mathrm{g}(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

Here b is called the bias or the offset value and is almost always set to 1. This is useful to maintain the activation of the neuron even when all the input signals are 0.

Some of the important concepts of neural networks that we would use are listed as follows:

2.1.1 Activation Function

While building the neural networks, we can choose the kind of activation function[24] we want to assign to different hidden layers in the network. The function $g(z)$ in the equation

1.1 is one such example of an activation function. The main purpose of using this activation function is so that non-linearity is introduced in the neural networks and the values are readjusted to a smaller range. In this way, depending on the threshold set, it can be concluded in a better way, if the neuron could be ”fired”, i.e., activated or not. There are different kinds of activation functions available, including sigmoid, tanh, ReLu etc.

Mostly, ReLu functions are used in the hidden layers and sigmoid function is used in the output layer due to their output range. Some of the activation functions can be represented as in the figure 2.2.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

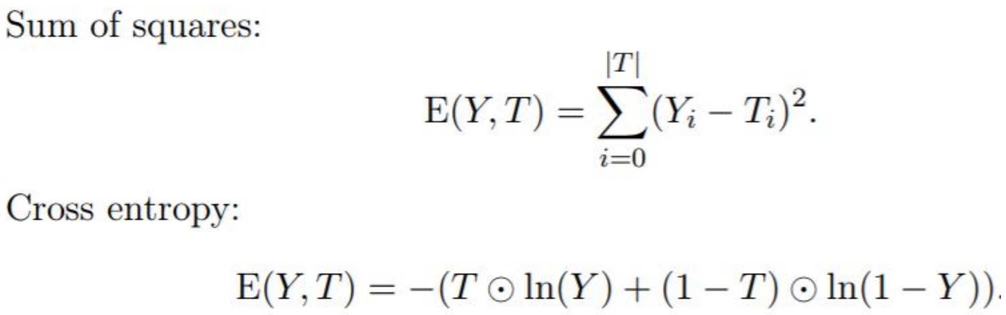
$$\mathrm{ReLU}(x) = \begin{cases} 0, & \text{if } n < 0 \\ x, & \text{otherwise} \end{cases},$$

$$\mathrm{LeakyReLU}(x) = \begin{cases} \alpha x, & \text{if } n < 0 \\ x, & \text{otherwise} \end{cases}.$$

Figure 2.2: A few of the widely used activation functions [2]

2.1.2 Back Propagation

When we arrive at the output layer of the neural network, we use loss function to see what the value of the error turned out to be. This is done by comparing the predicted value to that of the actual value that we expect. A few of the commonly used loss functions include sum of squares, Binary Cross-Entropy function[25], etc. They can be represented mathematically as seen in the figure 2.3.



Sum of squares:

$$\mathrm{E}(Y,T) = \sum_{i=0}^{|T|} (Y_i - T_i)^2.$$

Cross entropy:

$$\mathrm{E}(Y,T) = -(T \odot \ln(Y) + (1 - T) \odot \ln(1 - Y)).$$

Figure 2.3: Mathematical Representations of different Loss Functions [2]

The next step here is to calculate the derivative of this error value with respect to each weight in the existing neural network. To achieve this, Chain Rule of Differential Calculus is applied where the derivatives of the errors with respect to the values of the weights are applied to the final layer. These derivatives are known as gradient descents and these gradients are used to calculate the gradients of the earlier layers. This process continues till gradients of all the weights in the neural network are obtained after which the gradient value is subtracted from the weights. In this way, we use back propagation to obtain smaller error values.

2.1.3 Normalization

Normalization helps in bringing all the features in the model to a similar scale, which in turn helps in obtaining a more symmetric, circular cost function rather than an elongated, elliptical cost function that could have formed had the features not been normalized. This process helps in accelerating the training process by decreasing the number of steps for gradient descent to reach the local minima faster, thus eliminating the need to use smaller learning rate and improving it's speed.

One of the most widely used normalization methods is Batch normalization[26]. As the name suggests, this kind of normalization involves normalizing the inputs from the whole batch at a time and tries to limit the mean and variance values of the batch lie between 0 and 1.

$$y_{tijk} = \frac{x_{tijk} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}, \quad \mu_i = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_i^2 = \frac{1}{HWT}\sum_{t=1}^{T}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_i)^2.$$

Figure 2.4: Mathematical Representation for Bath Normalization. Mean (μ_i) and Variance (σ_i^2), values are computed as above, where T is the Batch Size, H is the height and $W$is the width[3].

The other kind of normalization technique that has proven to work well with Generative Adversarial Networks for Style Transfer applications is Instance Normalization[3]. This works a lot like Batch normalization, but instead normalizing the whole batch at once, it normalizes each of them individually.

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}, \quad \sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - mu_{ti})^2.$$

Figure 2.5: Mathematical Representation for Instance Normalization. Mean (μ_i) and Variance (σ_i^2), values are computed as above, H is the height and $W$is the width [3].

2.2 Convolutional Neural Networks

Convolutional neural networks[27] are a specific type of neural networks whose applications mostly lie in the field of computer vision. The input to the convolutional layers is almost always an image. Convolutional Neural Networks, most commonly known as CNNs use a smaller kernel or filter that is convolved with the image. In each step, the dot product of the corresponding cells of the image and the filter are computed and these values become the part of the feature map of the particular filter. The weights of this filter act as the parameters we plan to learn when a CNN is trained. Pictorial representation of it can be seen in the figure 2.6.

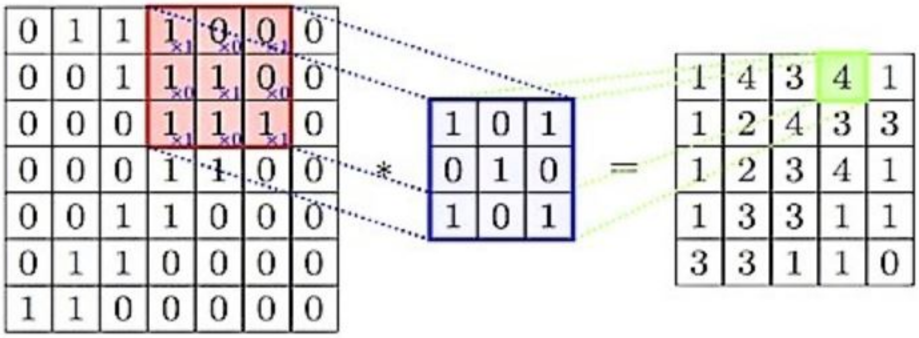


Figure 2.6: The process of convolution with a 3x3 filer [4]

2.2.1 Stride

In regular convolutions, the usual scenario is when a filter is applied and moved one step at a time on the previous feature map. In the above described case, the stride is 1 as only one step is taken incremented by the filter at a time. Therefore, stride can be defined as the number of pixels moved by the filter at each step. As the value of the stride increases, the size of the output keeps on decreasing proportionally.

2.2.2 Pooling

Pooling[28] layers are a different kind of layers used to reduce the size of the representation so that the process can be speeded up. There are two different types of pooling, namely Max Pooling and Average Pooling. In Max Pooling, each feature map of the input is operated by the pooling layer individually and the maximum value is passed as the output, whereas in Average Pooling, the average of the values is passed out as the output. Max Pooling is mostly used in most of the CNNs. Different kinds of Pooling are pictorially represented in the figure 2.7.

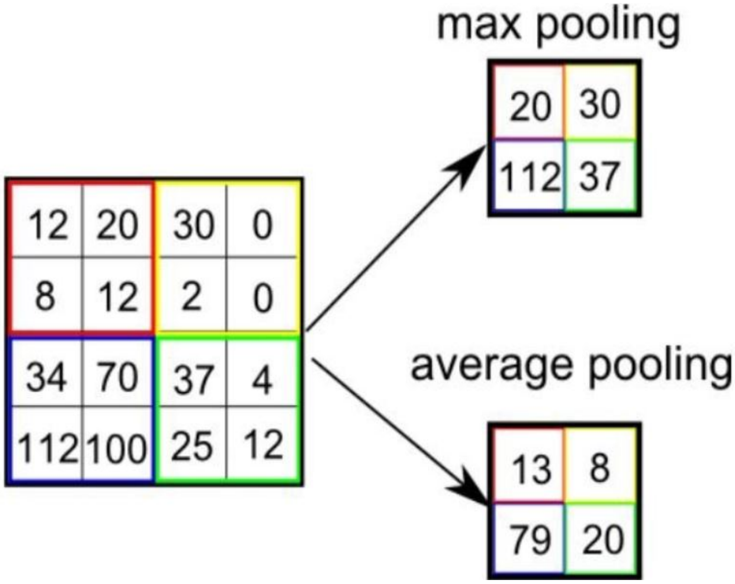


Figure 2.7: Types of Pooling Layers [5]

2.3 Supervised and Unsupervised Learning

Supervised learning is when we provide the model with input and want to map it to output labels or continuous output, as in the case of classification and regression respectively. In simpler words, it is the learning with input variables (x) and output variables (Y), and also makes use of an algorithm to learn the mapping from the input to the output and expects the model to learn and apply the transformation accurately when new input is given to it. Object classification is a good example to demonstrate Supervised Learning, as the training process learns the mapping using both the input and output values. Once trained, it tries to apply this learned mapping on to new examples.

In Unsupervised Learning[29], the model is given only input data and doesn't have any output data to map to. The main aim of Unsupervised Learning is for the model to learn the inherent structure or the distribution of the input data and apply it to the new unseen data once trained. The term ”unsupervised” is used as there is no mapping involved and the output has no correct answer. The algorithms or models are left to learn the required distribution by themselves and make their decisions on how to discover and interpret the structure within the data. Examples of algorithms that use Unsupervised Learning are Clustering, Auto-encoders[30], Generative Adversarial Networks[21], etc.

2.4 Generative Adversarial Networks

Generative Adversarial Networks[21] is a type of Machine learning model that is an example of Unsupervised Learning. Generative Adversarial Networks consists mainly of two deep neural networks, namely, the Generator network and the Discriminator network. These two networks are pitted against each other, and hence the term ”Adversarial” is used.

One of the most impressive characteristics of Generative Adversarial Networks, i.e., GANs is that they can mimic the distribution of the data they were provided as part of the training process. The Generator network creates new data instances, image instances in our case, whereas the Discriminator has to predict weather it is real or fake, i.e., if it is a sample from the original data or if it is generated by the Generator network. GANs output 0 for what they think is a fake image and 1 for a real image.

In traditional GANs, the Generator network learns how to map from a random latent distribution to the image distribution that resembles the training data. The discriminator which has access to the real data, makes a call whether the sample it receives is a generated one or is from the training data. Back propagation is applied to both the deep neural networks so that the Generator can produce better samples to fool the discriminator while the Discriminator can distinguish in a better way the fake image from the real one, i.e., generator's output to that of the real sample.

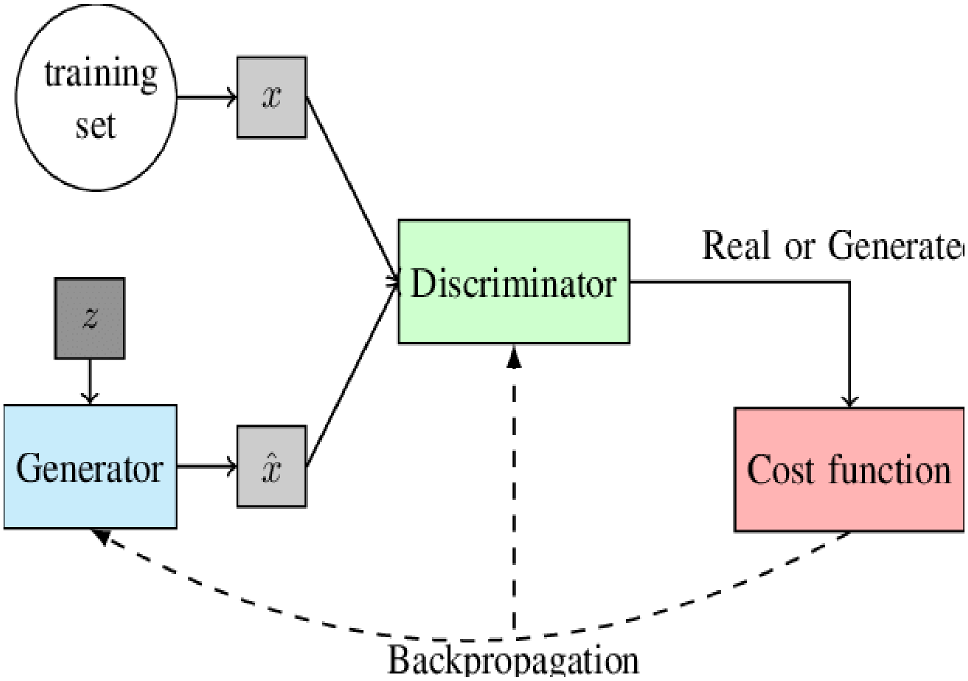


Figure 2.8: Working of Generative Adversarial Networks [6]

Here, both the Generator and the Discriminator play a mini-max game where the Generator tries to increase the error rate of the Discriminator by tricking it into believing that the samples are from the Generator while the Discriminator tries to reduce the same error rate. The worst case input for one networks is generated by the other network, and the adversarial training process includes the networks being trained on their worst case inputs. The cost function of the Discriminator and Generator can be seen in the following equations[21].

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p\text{ data}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log(1 - D(G(\boldsymbol{z}))) \tag{2.2}$$

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D(G(\boldsymbol{z})) \tag{2.3}$$

In the above equations, the Discriminator tries to minimize it's error rate $D(G(\boldsymbol{z}))$ whereas the Generator tries to maximize the same log probability of the Discriminator being mistaken. Mode Collapse is a phenomenon when a GAN is no longer able to generate all modes of image distribution as that target data. This happens when there is an image which minimizes the loss for the Generator and tricks the Discriminator, the Generator instead of going to different mode may learn to map every input to that point alone. This phenomenon can be detected when the output images all look the same instead of having varied distributions. This can be mitigated upto an extent using Mini-batch features approach described by Salimans et al 2016[31].

2.5 Related Work

The first stage of our proposed solution is to generate realistic looking images from the synthetic images generated through the 3D model that shares a similar background to our real images. Using conventional methods like DCGAN[29] was one of the approaches we wanted to try in the beginning but we later realized that DCGAN could not possibly help us over the issue of generating realistic images in different angles we aimed for, and neither is it robust enough to generate diverse range and number of images we desired to obtain through the GAN for the purpose of object detection in the later stages.

Though the initial instinct would have been to use a Generative Adversarial Networks, there have been many improvements in this field that pushed us to look at more advanced GANs that were capable of producing realistic images with better resolution. The final approach chosen, CycleGAN, build on the algorithm pix2pix which uses conditional generative adversarial networks to map the images from the input to the output. But CycleGAN was a more suitable approach to our problem as pix2pix[32] required paired training images in the corresponding datasets unlike CycleGAN which can learn from unpaired training examples.

Other developments in the field of Image Translation using GANs include CoGAN[33] which uses weight-sharing[34] technique in order to learn similar representations present across both the domains. Another interesting algorithm is SimGAN[35] which successfully generates realistic looking images from the synthetic refined images, much like our task here but it also relies on the input and output image data[35], [36], [37] to share specific content and use additional terms like class label space, image pixel space etc, which makes it harder to work with. CycleGAN does not have any of the constraints mentioned above and is also robust enough to generate as a huge number of images without facing the phenomenon of mode collapse easily.

Drone Detection problems and related questions[38] has been previously addressed before by researchers from University of Southern California[19] and few other universities[39], [40]. But all of them have made their own custom dataset and applied it on different object detection models according to their requirements and needs. They all have obtained the data for the training using some base station as reference which is a different viewpoint from what we wanted for our experiments. For our use case, we for the first time have supplemented real data with fake data for the purpose of drone detection.

Chapter 3

Solution Pipeline

In this chapter, the entire pipeline of the proposed solution is discussed which is condensed into two parts. The first part talks about choosing the appropriate method for the generation of realistic data, it's network architecture, the working of the algorithm etc. The second part of it deals with the downstream task of object detection, where we discuss reasons as to why the algorithm was chosen, working mechanism that it employs etc.

3.1 CycleGAN

3.1.1 Motivation

In this section, we discuss the reason for choosing CycleGAN for the purpose of generating realistic looking images. In the recent times, we have seen how Generative Adversarial Networks have successfully been applied to problems such as Style Transfer[41],[42], [43] Image-to-Image translation, which is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. Unlike the previous approaches discussed in the earlier chapter, the CycleGAN formulation does not rely on any predefined similarity function between the input and output, nor do it place any restrictions on the data that they must lie in a same low dimensional embedding space. All of the mentioned reasons, along with the fact that

CycleGAN uses unpaired training data makes it a very practical solution.

After a lot of research and examining the problem at hand closely, we have decided that in order to generate a number of realistic looking images, we need to have two datasets. One consisting of synthetic images rendered from the 3D Model and the other dataset that consists of real images from which synthetic images can learn the image distribution from. For this type of architecture, Image to Image translation is the most ideal approach where image distribution in one domain can be translated to image distribution in another domain. In the recent years, there have been dozens of papers on this concept but all of them required paired dataset of two different kinds, this meant that there should be pixel to pixel correspondence between the two datasets . This kind of data is hard to obtain and often is very small in number and obtaining this kind of paired data for this application is almost impossible.

As a solution, we are modifying CycleGAN which is one of the few algorithms where image distribution can be learnt between two datasets, one consisting of synthetic images and one consisting on the real images without the requirement of paired dataset. Our main focus is for the simulated images to learn from the real image distribution. The job of the discriminator is to differentiate the real image from the generated one and add realism to the synthetic so as to trick the discriminator.

3.1.2 Data Collection

To obtain images for the Real dataset, images are first gathered from the video taken by one drone capturing another drone in flight. The second drone was ripped off of all the extra circuits, to maintain resemblance between the 3D model and the actual drone. We mounted a GoPro 7 camera on one of the drones and used it to collect the necessary data. Later these pictures are zoomed in and cropped so as to get the close up images of the drone. All the drone images obtained had very similar background of the drone cage where we conducted our experiments. We collected around 3000 such images for the preparation of this dataset.

To obtain images for the synthetic dataset, we have chosen to render images from a 3D Drone Model that was purchased online. The 3D model is of the drone, DJI Matrice 100 which is the exact drone model we used to conduct our experiments. Efforts were made to edit the 3D model for it to look very similar to the actual drone we were using. The images from the 3D model were rendered using the Autodesk Maya software. The camera was set up in such a way that it could capture the drone in a 360 degree circle using a turntable animation. This was done at different elevation angles so that we had pictures of the drone at various angles and positions. The background image used was HDRI in nature so that realistic illumination conditions could be produced. The background in the synthetic images is intentionally kept similar to the background we see in images of the real dataset so that CycleGAN model does not have to put in a lot of effort in translating the real image distribution to the images in the synthetic domain. Around 3000 such images were generated to make this dataset.

3.1.3 Algorithm

CycleGAN is a Generative Adversarial Network that is designed for image-to-image translation of unpaired images, where the main objective is to translate images from one source domain X to another target domain Y. The main objective for this algorithm is to learn the mapping between two image collections, rather than between two specific images, by trying to capture correspondences between higher-level appearance structures.

The goal is to learn a mapping $G : X{\rightarrow}Y$, such that the distribution of images from G(X) is indistinguishable from the distribution Y using an adversarial loss. Because this mapping is highly under-constrained, we couple it with an inverse mapping $F : Y{\rightarrow}X$ and introduce a cycle consistency loss to enforce $F(G(X)) \approx X$ (and vice versa).

The CycleGAN architecture consists of two Generative Adversarial Networks, one for translating from domain X to Y and another for translating from domains Y to X. The model consists of two mapping functions through the generators :

$$G_X : X \rightarrow Y \tag{3.1}$$

and

$$F_Y : Y \rightarrow X \tag{3.2}$$

Along with the generators, there exist two adversarial discriminators of the two GANs, namely, D_X and D_Y , where D_X aims to distinguish between images $\{x\}$ and translated images $\{F(y)\}$; in the same way, D_Y aims to discriminate between $\{y\}$ and $\{G(x)\}$. These can be represented with the below functions :

$$D_X : X \rightarrow \mathbb{R} \tag{3.3}$$

and

$$D_Y : Y \rightarrow \mathbb{R} \tag{3.4}$$

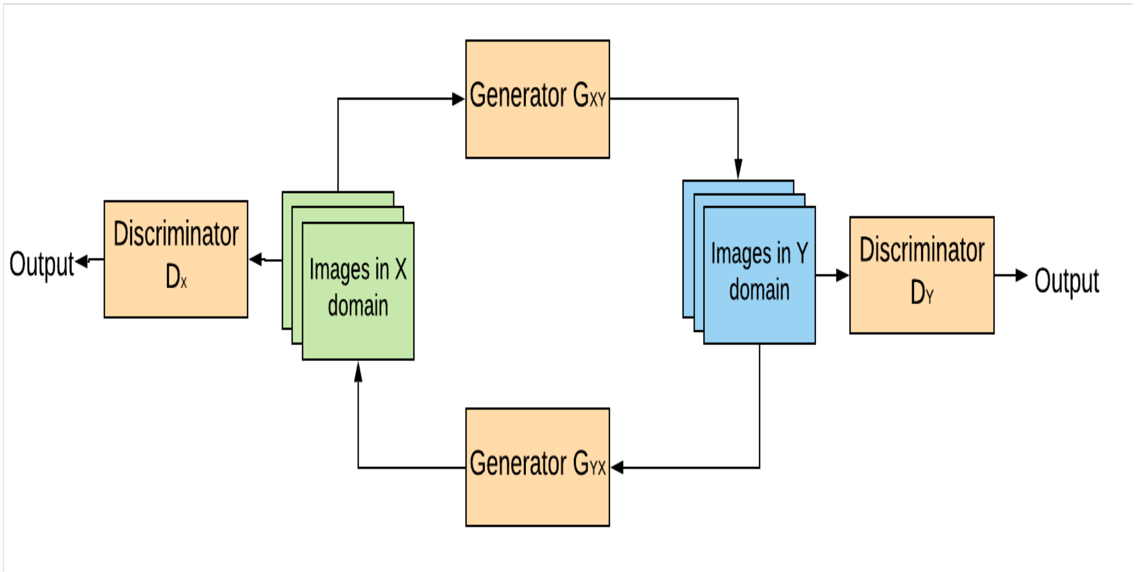


Figure 3.1: High Level working of the CycleGAN [7]

3.1.4 Role of Generators

The generator G_X maps the images from domain X to domain Y. The images generated by G_X are mapped back to the original domain by F_Y. The other generator F_Y should make sure that the image generated by it should accurately represent the original image that was the input of G_X. Here the concept of the cycle-consistency loss is introduced[35] which helps in minimizing and optimizing the distance between the actual input image given and the regenerated or reconstructed input image.

3.1.5 Role of Discriminators

The discriminator D_X encourages the generator, G_X to translate images from domain X to produce outputs that can not be distinguished from domain Y, and vice versa. The role of the discriminator D_X is to discriminate between the images of the domain X and translated images $F(Y)$.

There are mainly two types of losses we encounter, adversarial loss and cycle consistency loss. Adversarial losses is used for matching the distribution of generated images to the data distribution in the target domain; and cycle consistency losses to prevent the learned mappings G and F from contradicting each other.

Adversarial Loss for the Generator X [44], G_X can be defined as :

$$\mathcal{L}_{\text{ GAN }}\left(G, D_Y, X, Y\right)=\mathbb{E}_{y \sim p_{\text{ data }}(y)}\left[\log D_Y(y)\right] \quad+\mathbb{E}_{x \sim p_{\text{ data }}(x)}\left[\log \left(1-D_Y(G(x))\right]\right. \tag{3.5}$$

where $\mathbb{E}_{y \sim p_{\text{ data }}(y)}\left[\log D_Y(y)\right]$ is the loss obtained for the original images from the domain Y and $E_{x \sim p_{\text{ data }}(x)}\left[\log \left(1-D_Y(G(x))\right]\right.$ is the loss for corresponding to the fake generated images produced by generator G_X.

In this case, the generator, G_X tries to generate images $G(X)$ which share similarity to the images corresponding to the domain Y , while D_Y focuses on discriminating between the transformed images $G(X)$ and real samples y. In a nutshell, the generator, G_X tries to minimize this objective against the adversary D_Y that tries to maximize it, using the function $\min_G \max_{D_Y} \mathcal{L}_{\text{ GAN }}\left(G, D_Y, X, Y\right)$, thus playing a minimax game amongnst them.

In the same way. a similar loss function can be calculated for the other generator F_Y and $\min_F \max_{D_X} \mathcal{L}_{\text{GAN}}\left(F, D_X, Y, X\right)$.

It is known that adversarial training can make it possible to learn mappings G and F from their respective domains, and can also generate images which are as identically distributed as the original domains X and Y. Even when this is the usual scenario, with a huge capacity, it is possible for the model to map one input image or a set of input images to any random permutation or combination of images in the respective target domain, where any trained learned mappings can produce an output image distribution which is similar to required target domain distribution.

Because of the above reason, it can be safe to say that adversarial losses alone cannot ensure that a learned network can map a single image x_i in the domain X to a similar looking image in the domain Y to produce the respective image y_i. To further overcome and reduce the space of other possible image mapping functions, it was argued that the learned mapping functions should be cycle-consistent.

The concept of cycle-consistency can be explained through the following pictures where a for particular image x_i of domain X, where mappings G and F are applied consecutively on in, i.e., when image translation cycle is applied the original image x_i will be produced. Mathematically, this can also be understood as : $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. This was called forward cycle consistency. This can be observed in the figure 3.2.

Similarly the illustrated figure 3.3 shows that for every image in the domain Y, G and F mappings must also satisfy the backward cycle consistency law where $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$.

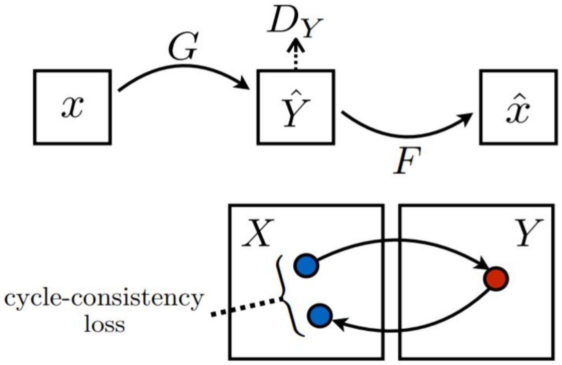


Figure 3.2: Illustration of Forward cyclic loss[8]

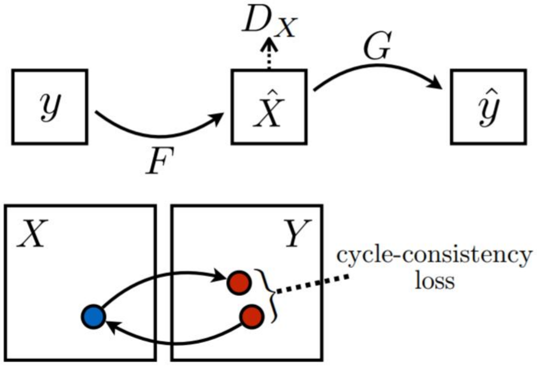


Figure 3.3: Illustration of Backward cyclic loss[8]

This cycle consistency loss can be mathematically represented as :

$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G,F) = & \mathbb{E}_{x\sim p_{\text{data}}(x)}[\|F(G(x))-x\|_1 \\ & +\mathbb{E}_{y\sim p_{\text{data}}(y)}[\|G(F(y))-y\|_1]\end{aligned} \tag{3.6}$$

The full objective function of CycleGAN can be written as :

$$\begin{aligned}\mathcal{L}(G,F,D_X,D_Y) = & \mathcal{L}_{\text{GAN}}(G,D_Y,X,Y) \\ & +\mathcal{L}_{\text{GAN}}(F,D_X,Y,X) \\ & +\lambda\mathcal{L}_{\text{cyc}}(G,F)\end{aligned} \tag{3.7}$$

Where the relative significance of the above two objective functions is dictated or decided by the value λ.

The CycleGAN is compared against the objectives that were formed by changing the full-objective function, such as only taking into consideration the adversarial loss, or taking only cycle-consistency loss into account etc. Studies showed that both the objectives or losses have a crucial role in delivering the best quality results that were witnessed through CycleGAN. It was therefore concluded that both the objectives were an important part of the objective function to get good results.

Apart from the above changes, it was also seen that training the CycleGAN in only one direction did not produce good results. It was observed that when trained with the forward loss alone like in the case of Generative Adversarial Network + forward Cycle-loss, i.e., $\mathbb{E}_{x \sim p \text{ data }(x)}\left[\|F(G(x))-x\|_1\right]$, the training was not stable and often caused the problem of mode-collapse, mainly in the direction that was not used for training, in this case, the backward loss. Similarly results were observed when Generative Adversarial Network+Backward cycle-loss were used, i.e., $\mathbb{E}_{y \sim p \text{ data }(y)}\left[\|G(F(y))-y\|_1\right]$.

3.1.6 Network Architecture

In this section we will go over the architecture adopted for the Generator and Discriminator in the CycleGAN model. In CycleGAN, two inputs are fed into each discriminator. One input is the original image from domain X and the other image is the transformed image through the generator X to Y. In both the cases, the role of the discriminator is to distinguish if it is a real or a fake(generated) image so that it can trick the generator and recognize images generated by it. At the same time, each Generator aims to generate images that

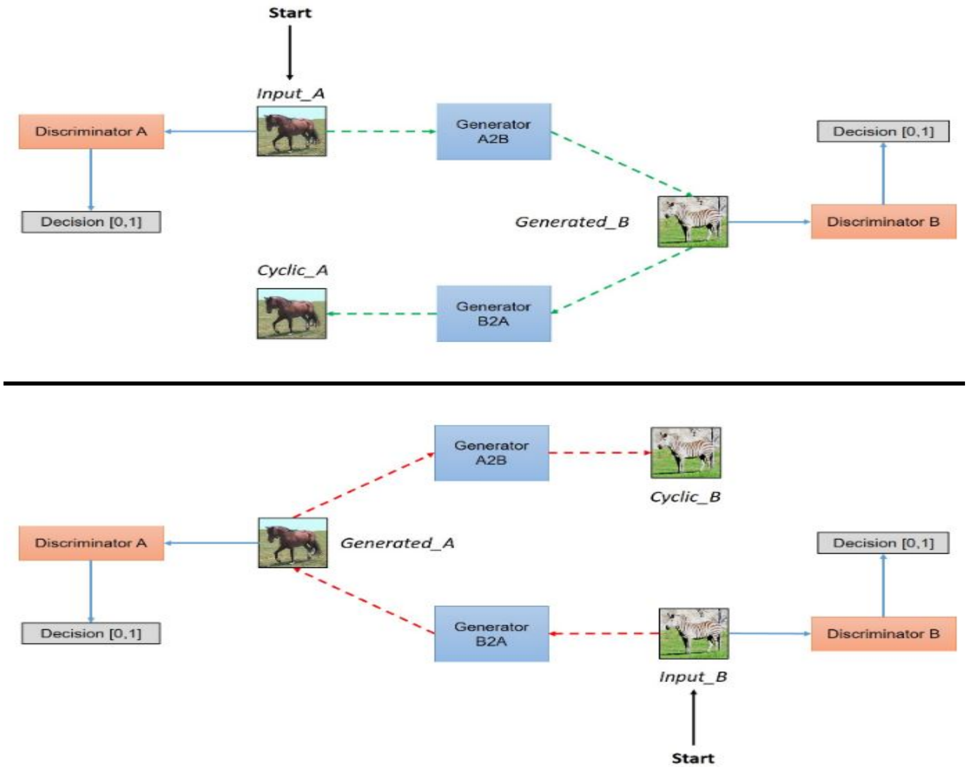


Figure 3.4: Illustration of the working of CycleGAN [9]

closely resemble images from the original domain. In this way, both the Generator and Discriminator play this game to attain equilibrium so that the distribution of each generator is that of the target domain distribution. This process can be seen in the figure 3.4.

3.1.7 Generator Architecture

The network architecture of the Generator network has been heavily inspired from Johnson et al[41], who have displayed impressive results for their work involving Perceptual Losses for Neural Style Transfer and Super-Resolution. The architecture of the generator can be seen in the figure 3.5.

The architecture of CycleGAN comprises of layers that involve three levels of computation. The first level comprises of series of three convolutional layers which are used for

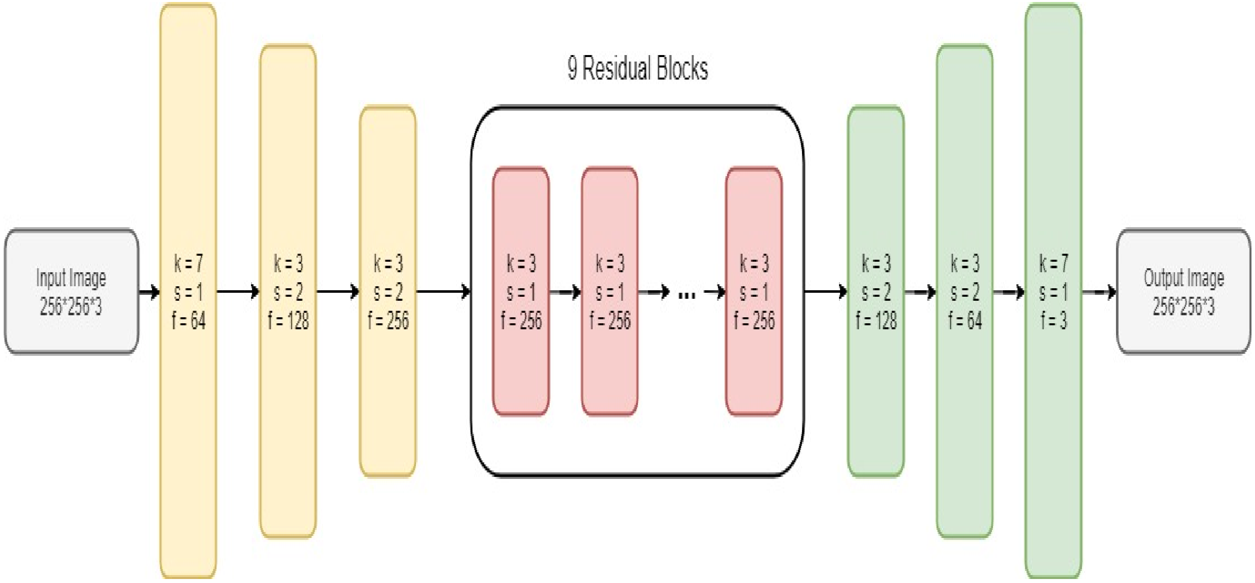


Figure 3.5: The convolutional layers have the kernal size of 3x3 respectively. Here k represents the kernel size, f represents the number of filters, and s describes the stride value. The above architecture is for the image size of 256x256 and its consists of nine residual blocks.

downsampling of images. The three layers help in extracting the image features, here the representation size keeps decreasing while the number of channels keeps increasing. The resulting activation is then passed to the next level, where it helps in transforming the features from the input domain to the target domain. A series of nine connected residual blocks is responsible for this transformation.

The resnet blocks comprise of ResNet layers[45] which come from Residual Networks. Each of them consist of two convolution layers where the residual input is added to the output. The intuition behind this is so that the performance of the ResNet block would not be worse than the identity mapping as the availability of the input is always available. In this way, the properties of the previous input layers is present for the future layers as well. They try to retain the characteristics of input objects like size, shape etc in the output images so that there won't be any drastic deviations from the input images. Moreover, the residual networks also help in alleviating the gradient vanishing problem for deep networks. The

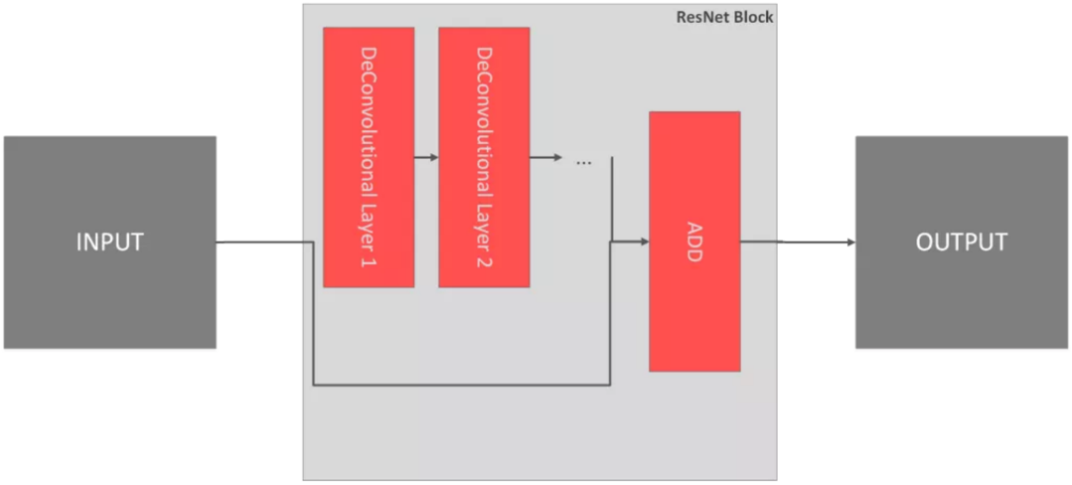


Figure 3.6: ResNet Architecture [10]

architecture of the ResNet can be seen in the figure 3.6.

In the later stage, upsampling of the images is carried out to enlarge the representation size. Here two deconvolution layers are used to decode the transformed features to obtain the final output image which would be of the same size as that of the input image. This can be seen clearly in the figure 3.5. As the architecture of the generator network is completely convolutional, it hence would be equipped to handle arbitrarily large input once the model is trained.It should be noted that in the figure 3.5, every layer is followed by instance normalization and a ReLU layer.

3.1.8 Discriminator Architecture

The architecture of the discriminator network has been adopted from PatchGAN[32], [46] which was described in the pix2pix paper. In order to restrict attention to the structure in local image patches, rather than classifying the whole image, PatchGAN only computes if NxN patches of the images are real or fake. The PatchGAN discriminator are completely

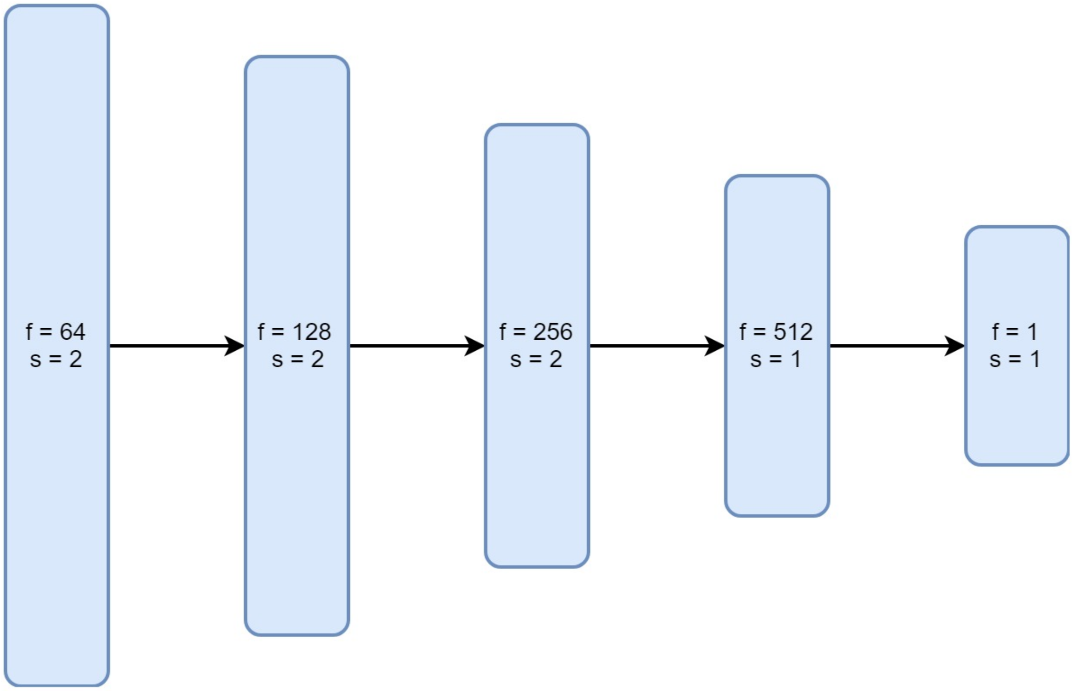


Figure 3.7: This is the PathGAN Discriminator Architecture. The filter size is 4x4, ”s” denotes the stride value and ”f” is the number of filters used. The input to it is an image and the output is the decision weather the input image is fake or real.

convolutional neural networks that just look at a patch of image, and determines the probability of the it being real or fake. The architecture of it can be seen in the figure 3.7. This works computationally better as PatchGAN now has to deal with comparatively lesser number of parameters and in a way, it is also faster than classifying the whole image. This also is more effective and helps in giving better results as more focus can be put on high level features like texture, etc. After testing it in various experiments, the authors determined that 70x70 patch size was able to give the best results.

One training modification that needs attention is that the discriminator is trained on a previous history of batch of generated images rather than just the previous generated image.

This is a technique first employed in the SimGAN paper[35] where they saw an improvement in the stability of training. An image buffer of size 50 is used for this purpose that stores the last 50 generated images to update the discriminator. The code for this method is used from the publicly available official github repository [44].

After training the CycleGAN model with the two datasets as input, the model is now ready to transform any number of images from Domain X to Domain Y. These images will look quite similar to the target distribution which consists of real data.

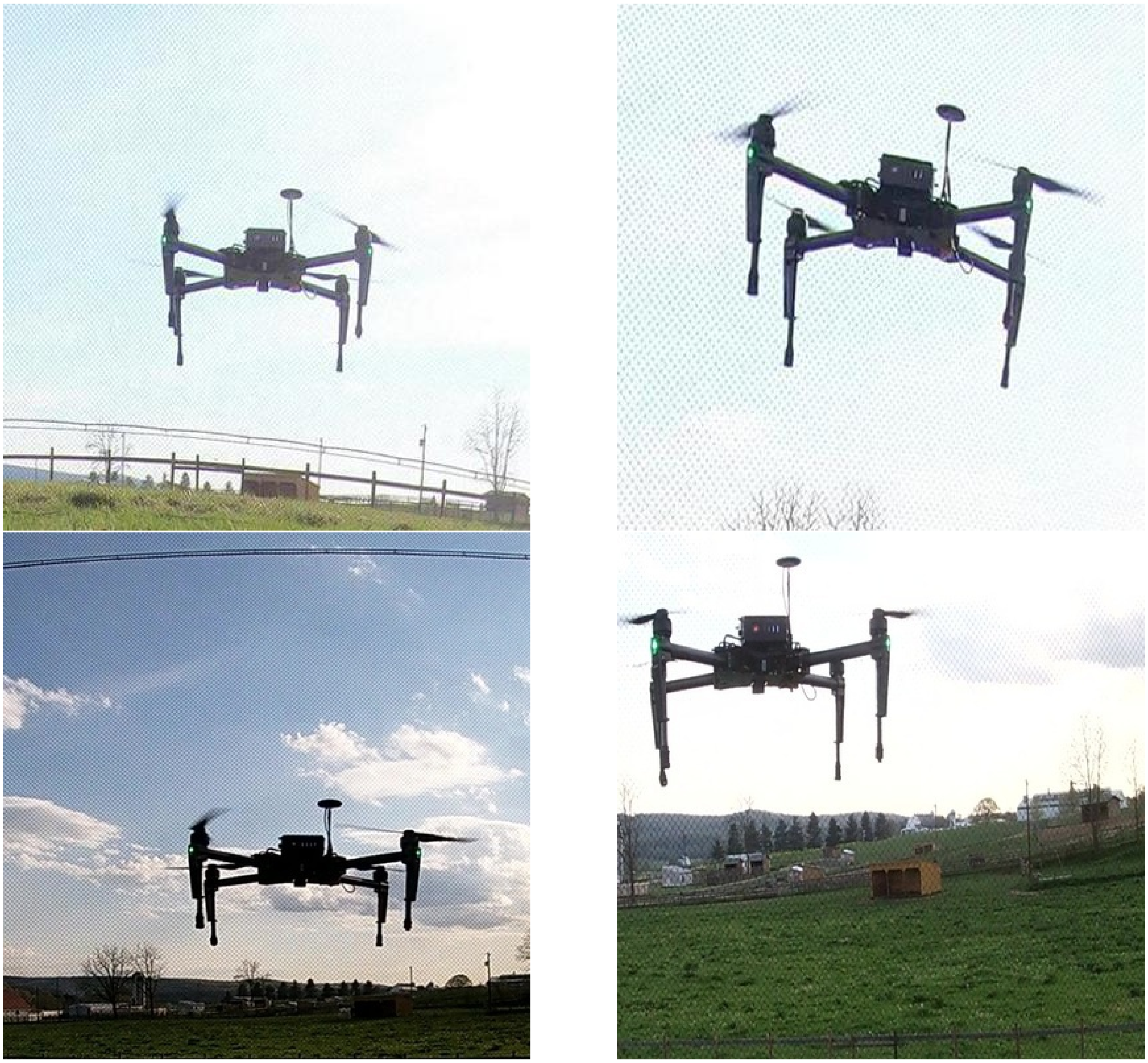


Figure 3.8: A few images from the Real Dataset

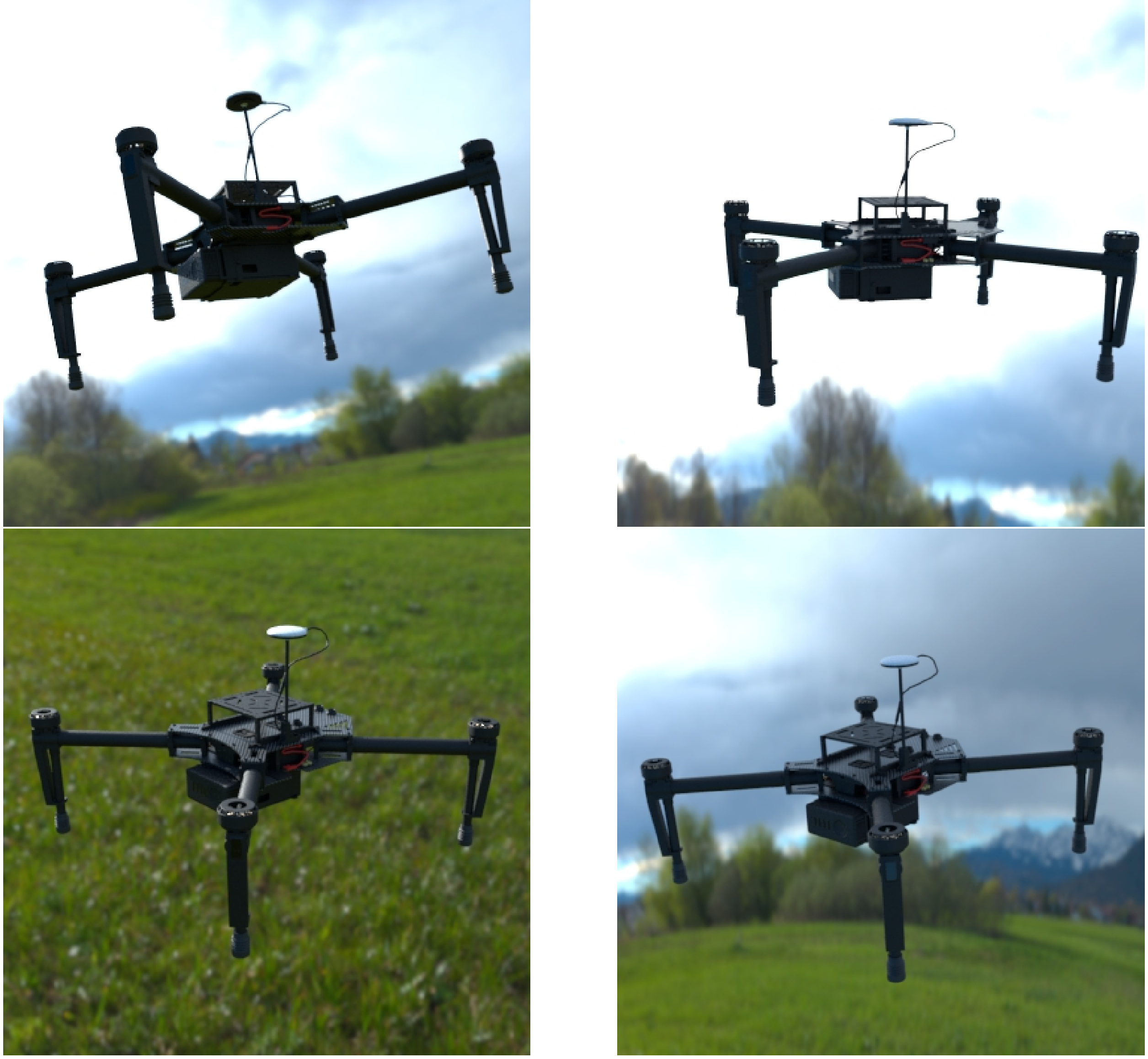


Figure 3.9: A few images from the Synthetic Dataset

Above are some of the sample images from Real dataset and Synthetic dataset. We aim to transform synthetic data to look like real images so that we can obtain any number of realistic looking images we want, from any perspective we choose.

In our case, the next step in the pipeline is to obtain the generated images from the CycleGAN, and use these images to train the drone detection model. The algorithm that we chose for this purpose is YOLO, i.e., You Only Look Once Algorithm which is explained in the next section.

The entire pipeline of the complete process can be seen in the figure 3.10.

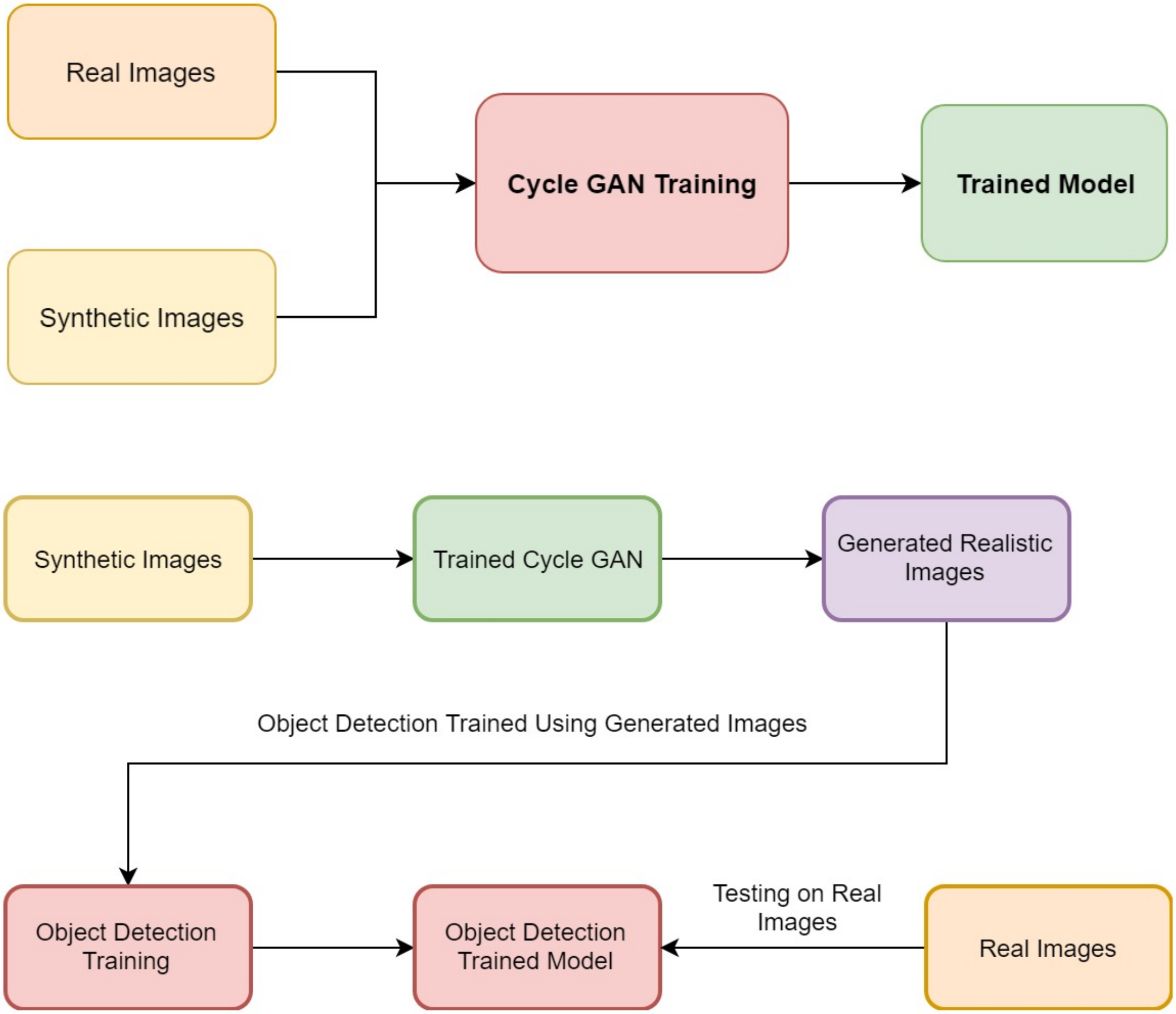

Real Images
Synthetic Images
Cycle GAN Training
Trained Model
Synthetic Images
Trained Cycle GAN
Generated Realistic Images
Object Detection Trained Using Generated Images
Object Detection Training
Object Detection Trained Model
Testing on Real Images
Real Images

Figure 3.10: In our case, the next step in the pipeline is to obtain the generated images from the CycleGAN, and use these images to train the drone detection model. The algorithm that we chose for this purpose is YOLO, i.e., You Only Look Once Algorithm which is explained in the next section.

3.2 Object Detection : YOLO

3.2.1 Introduction

Object Detection is the process of identifying one or more type of objects in a picture or video and specifying its location with the help of drawing a bounding box around the identified objects. YOLO, i.e., You Only Look Once algorithm is one of the fastest and accurate real-time object detection algorithms that are available. For this reason we have chosen the state of the art YOLO algorithm for the purpose of detecting drones in our datasets.

The first version of YOLO was introduced in the year of 2015. Since then, there have been quite a few improvements on the initial algorithm and currently we made use of the latest version of YOLO, i.e., YOLOV3[14].

Earlier object detection algorithms repurposed localizers to perform detection of objects. Those algorithm was applied to an image at multiple location with varying scales[47] to detect the presence of the object they were looking for and depending on the probability scores of the bounding box in each image region, the final bounding boxes were constructed and object detection was carried out this way.

YOLO[20] on the other hand uses a completely different approach to detect objects. In this method, only a single neural network is applied on the entire image. This network then divides this image into different grid like regions and predicts bounding boxes for each with their corresponding probabilities. When compared to the other detection algorithms that use region proposal networks like Fast RCNNs[48], etc, which perform detection on various region proposals and end up performing detection multiple times on various scales of the image, the architecture of YOLO is a lot more efficient as it is like a Fully Convolutional Neural Network. The image gets passed into the network once and gives out the image with the corresponding prediction.

3.2.2 Working of YOLO Algorithm

In the algorithm, the system first takes the input image and divides in such that it becomes an N × N grid. Each grid is responsible for detecting the midpoint of that object. In the scenario where the midpoint of the object is present in a particular grid, that grid would be responsible for the detection of that object. The output of every image comprised of a bounding box that makes five predictions which include the x, y, w, h coordinates and the confidence score or the probability of the object being a drone. This can be seen in the figure 3.11.

The coordinates x and y describe the center of the grid relative to it. The width and height are predicted relative to the whole grid cell. They can sometimes be below the value 1.0, and if the object crosses the bound of the grid, they can be of values greater than 1.0 , this is the the border of the bounding box goes beyond the grid region. In YOLO, each cell was capable of drawing 2 bounding boxes, this number went up till 5 later in the YOLOv2 stage. As part of one of the outputs, the confidence score is also mentioned which is based on the IoU score using the relationship between the predicted bounding box region and the ground truth established. This score helps in indicating the probability by which the model thinks that the highlighted object is a drone.

The confidence scores of the object detected talks about how confident and accurate the model is about the detected object being a drone. In a more deeper sense, confidence score

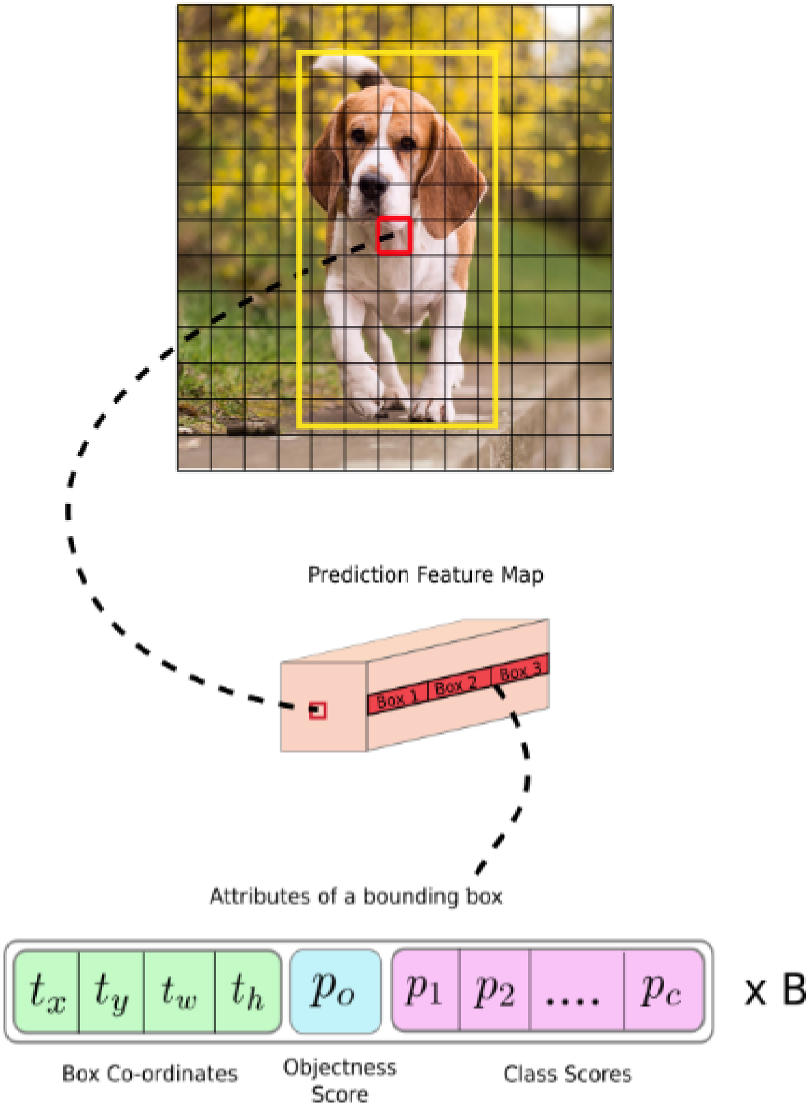


Figure 3.11: As described in the figure above, the YOLO algorithm first divides the image into an N x N grid first. With each grid is associated a feature map, which talks about all the classes it has identified, the bounding box coordinates and the its corresponding confidence scores. The above image is that of a 19 x 19 grid and each cell has the capability of predicting 5 bounding boxes each, which would sum upto a total of 1805 bounding boxes for one image. [11]

is the Pr(Object)*IoU score. If the object sees no object in the bounding box, the confidence score associated will be 0 and there would not be any coordinates detected. This can be demonstrated in the figure .

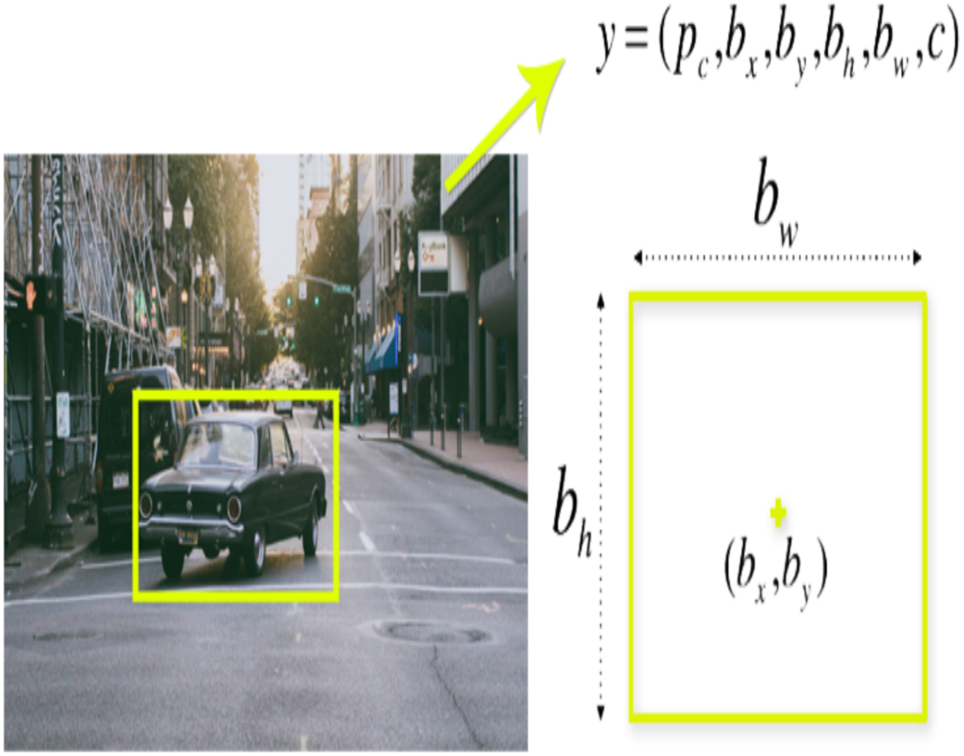


Figure 3.12: Bounding box representation [12]

The next step of the algorithm consists of a certain step that deal with something called Non-max suppression. It sometimes may happen that one may find multiple bounding boxes around one object as it might have been detected multiple times. This could be possible due to the reason that more than one grid cell might think that it possess the center of the same object. So for this reason instead of identifying it just once, multiple bounding boxes might appear identifying the same object multiple times as shown in the figure below. Non-max suppression, as the name suggests is one of the ways to ensure that only once is the object detected.

In the figure 3.13, if a 19 by 19 grid is places on it, it can be seen how multiple cells would think that the mid point of the car in their grid and each of them predict the presence of the car. In practice, the object classification and localization task must be running for each of these grid cells. In this way, we see how multiple detections are possible for the same image. Non-max suppression cleans and eliminates the extra detection bounding boxes such that each object would end up having only one bounding box with highest confidence score. [h] The algorithm first looks at the probabilities associated with each of these detec-

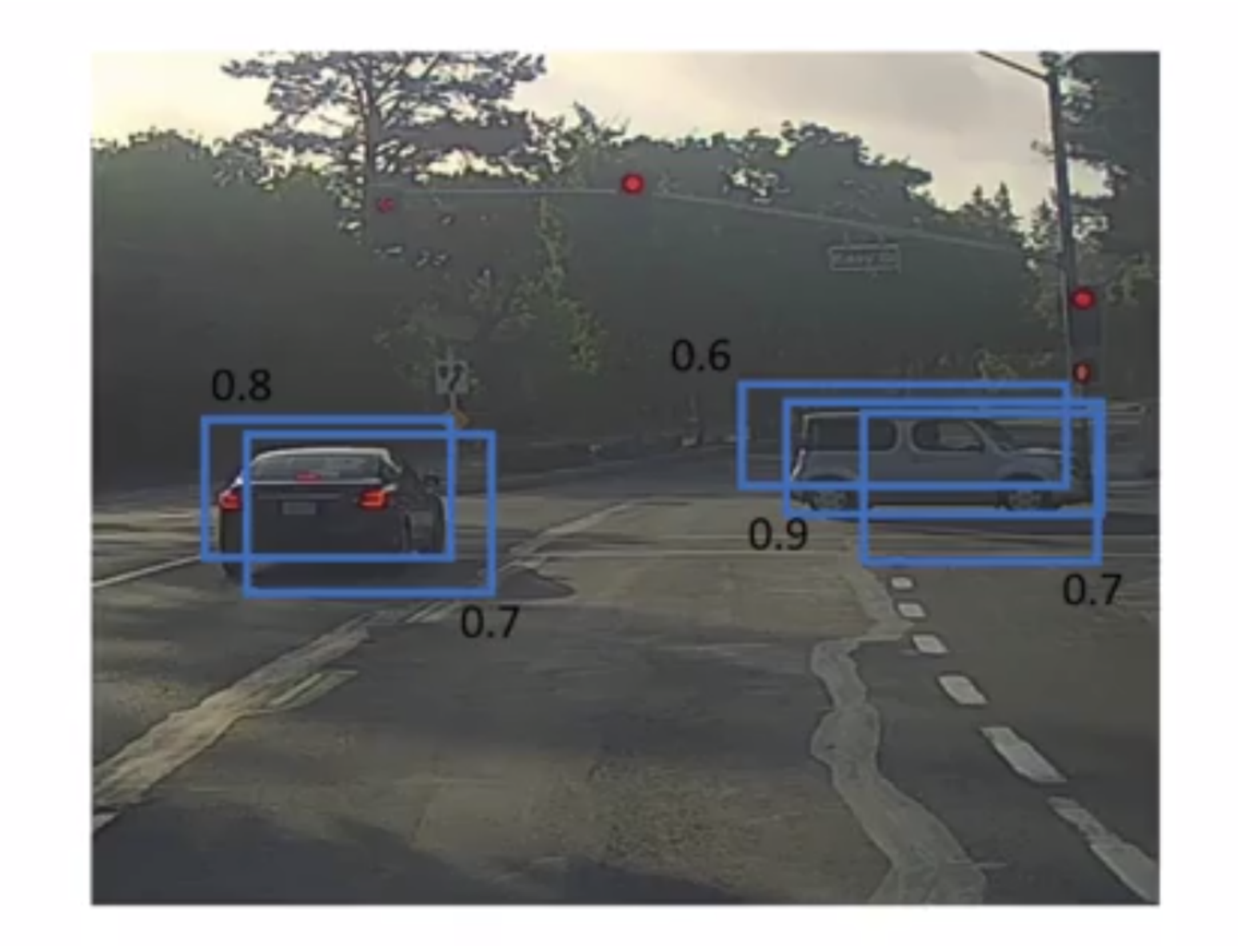
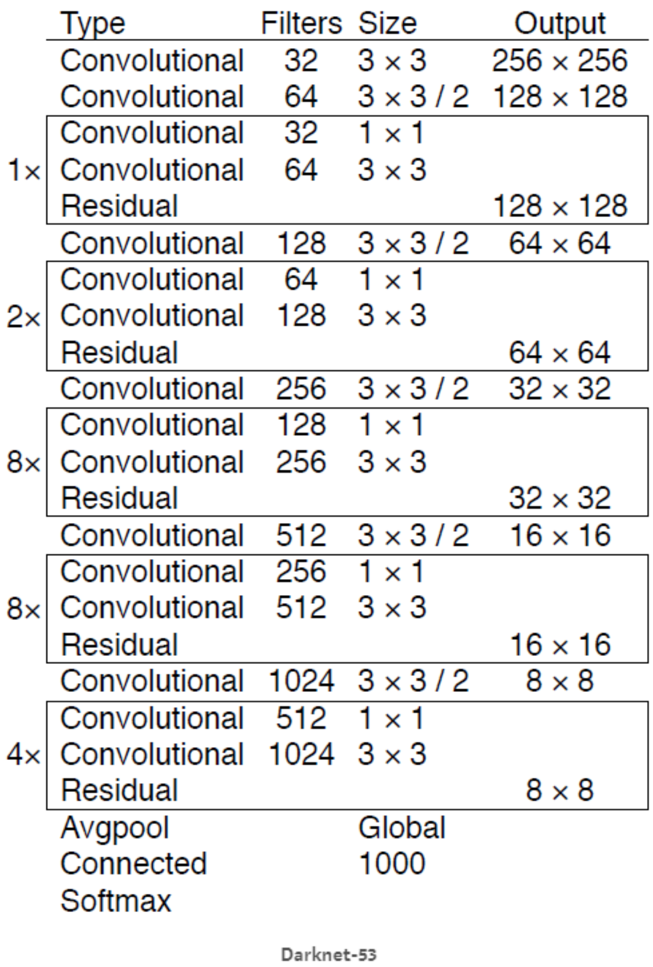


Figure 3.13: Presence of Multiple Bounding Boxes before Non Max Suppression[13]

tions. Through a series of steps, it essentially eliminates the bounding boxes with low object probability and the bounding boxes that share the highest IoU with the bounding box with highest probability score so that one object would have only one bounding box. This process is called Non-max suppression where that the bounding boxes with maximum probability would be sent as the output but the near by ones, that are not of that great probability would get suppressed.

The first YOLO algorithm was a fully convolutional network with about 24 convolutional layers followed by 2 fully connected layers. However over time, the architecture of the YOLO network keeps changing and the for the present YOLOv3 version, the network architecture for this looks like the image below.YOLOv3 includes improved architecture which includes most important elements like ResNets, skipping connections, upsampling etc. YOLOv3 now uses Darknet-53 which means that it uses 53 convolutional layers in the network architecture. Both YOLOv2 and v3 use batch normalization.

The latest proposed network architecture of latest YOLO algorithm, i.e., YOLOv3 can be seen in the figure 3.14.



	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			

Darknet-53

Figure 3.14: Network Architecture of YOLOv3 [14]

3.2.3 Loss Function

Previously, the loss function of YOLO v2 looked something like the following :

$$\begin{aligned}
&\lambda_{\text{ coord }} \textstyle\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&\quad + \lambda_{\text{ coord }} \textstyle\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&\quad + \textstyle\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&\qquad + \lambda_{\text{noob}j} \textstyle\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&\qquad\quad + \textstyle\sum_{i=0}^{S^2} \mathbb{I}_{i}^{\text{obj}} \sum_{c \in \text{ classes }} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned} \tag{3.8}$$

YOLO made use of the sum squared loss to calculate the error loss between the predictions made and the ground truth declared.

The first two terms represent the localization loss which describes the errors in the predicted bounding box coordinates and its sizes. To be more precise only the bounding box chosen to represent the object is held responsible for this loss. $\mathbb{I}_{ij}^{obj}$ is equal to 1, if jth bounding box is responsible for detecting the object in cell i, else it is equal to 0. As it is not fair to weigh the absolute errors in the large and small bounding boxes equally, the authors take the square root of bounding box width and height as it can be seen in the above equation.In order to increase the weight of the loss in the bounding box coordinates and put more emphasis on the precision of the boundary box, the loss is multiplied by λ_{coord}.

The next two terms represent the confidence loss in the above equation which can be described as the measure of the presence of the object in the box. Also, it should be kept in mind that not every grid cell contains an object in it, which makes their confidence score move towards 0, which leads to the domination of gradient from grid cells that do not contain objects. in every image many grid cells do not contain any objects. To balance this problem, the loss belonging to the bounding box coordinate predictions is increased and the confidence prediction loss for the bounding box that does not contain an object is decreased.

The two terms used here are λ_{coord} of default value 5 and λ_{noobj} of default value 0.5.

The last term in the loss function is the classification loss which is the squared error loss of class conditional probability for each class in every grid of the image. The term $\mathbb{I}_i^{\text{obj}}$ is 1 when object appears in the cell, or it is 0.

The above loss function is the one that was used previously by YOLOv2, but in the latest version of YOLOv3 the last three terms which were the squared errors, were replaced by cross-entropy error terms. This means that the object confidence loss and the class predictions loss functions in YOLOv3 are now calculated using logistic regression.

In the revised version of YOLO v3, to detect images it replaces previously used softmax function and performs multi-label classification for detecting the objects. Here, the class score for each class is predicted with the help of logistic regression and a threshold is assigned to it to predict multiple labels for a object. Classes which have higher threshold value are then assigned to the that particular bounding box. We used the code for Object Detection model from Alexey's github repository[15].

3.2.4 Evaluation metrics

In this section, we will go over the metrics used to define the results we obtain during the testing phase after the training process is complete. The main metrics that were made use of to make few important observations are the following.

3.2.5 IoU (Intersection over Union)

Intersection over Union is one of the evaluation metrics used to measure how accurate the object detector is. As the name describes, it is the ratio of the are of the overlapped region between the predicted bounding box and the ground truth bounding box to the union of the areas of their respective bounding boxes. This can be understood in a better way with the figure 3.15. To use this metric on a object detector model, there are two things one would need :

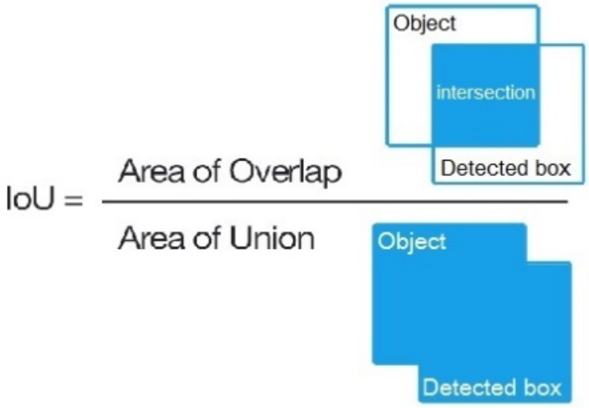


Figure 3.15: Representation of IoU[15]

- Ground-truth bounding boxes : These are the hand labelled bounding boxes over the images in the dataset hwere the location of the object, in our case, drone, is provided.
- Predicted bounding boxes that the object detection model generates.

This is an excellent metric that describes the accuracy of the bounding boxes drawn in our test or validation dataset. Generally, predicted bounding boxes that overlap heavily with the ground truth bounding boxes are considered more accurate. The pictorial representation

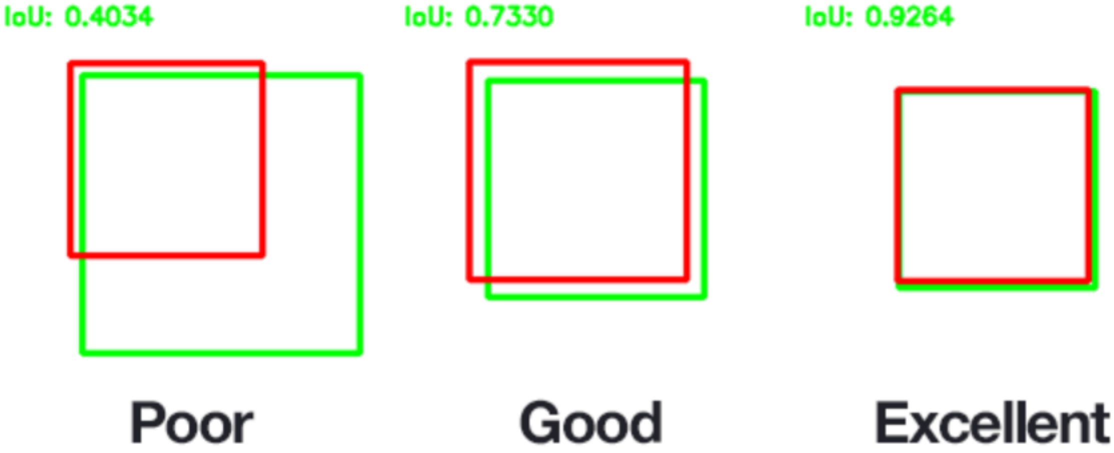


Figure 3.16: Relationship between the bounding boxes and IoU values[16]

of the same can be seen in the figure 3.16.

3.2.6 Precision

Precision is a metric used to measure how accurate the predictions turned out to be, i.e., it can be described as the percentage of correct predictions made out of all predictions. It can be seen as the ratio of number of True Positives to that of all the elements that were labeled as belonging to the positive class.

Here, false positives are the elements that were labelled incorrectly and true positives are the elements that actually belong to that certain class and labelled correctly.

3.2.7 Recall

Recall measures how well all the positives were found. Recall can be defined as the ratio of number of positives that were correctly identified to that by the total number of existing relevant elements that belong to the particular class. In this scenario, it can be is defined as

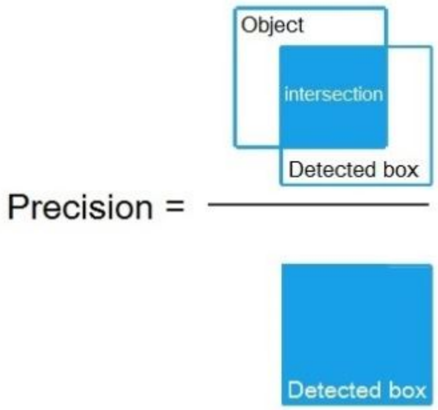


Figure 3.17: Precision[15]

the number of true positives divided by the total number of elements that actually belong to that particular class, which might include elements that were not labelled correctly to this class.

Here, false negatives are the elements that were in fact a part of the labelled class, but were

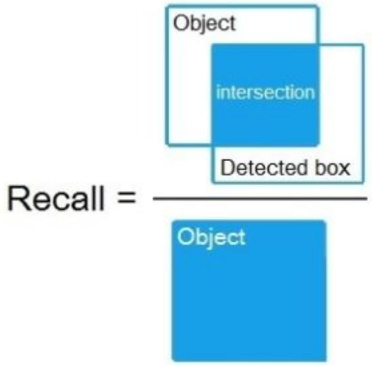


Figure 3.18: Recall[15]

not taken into consideration as they were incorrectly labelled. Below are the equations that explain precision and recall in a better way.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

TP = True positive

TN = True negative

FP = False positive

FN = False negative

Figure 3.19: Calculation of Precision and recall Values [17]

3.2.8 mAP (Mean Average Precision)

:

Before we understand mAP scores, we need to plot Precision-Recall curves, with precision on the Y axis and Recall on the X-axis. The precision-recall curve is computed from the model's detection output, by varying the model score threshold that determines what is counted as a model-predicted positive detection of the class. The final graph would look similar to that of the above picture.

The next step to calculate the Average Precision score is to take the average value of the precision across all recall values. The general and a more broader definition for the Average Precision (AP) is finding the area under the precision-recall curve. As both Precision and recall values are ratios, their values are always going be less than 1. Therefore, the Average precision, which we define to be the area below the curve is also always between 0 and 1. Each unique point is used on the Precision Recall curve to calculate the area it covers without any approximations. This definition is also called the Area Under Curve (AUC).

$$\text{AP} = \int_0^1 p(r)dr \tag{3.9}$$

Therefore, mAP (mean average precision) is the average of AP for all the classes or categories. In some context, AP is first computer for all classes and then averaged. This metric is one of the most important metric that speaks about how accurate the object detection model is performing.

Chapter 4

Experiments and Results

In this chapter, the experiments carried out and their results obtained are discussed in detail. Let us go through the CycleGAN experiments first followed by the experiments conducted and the results obtained for YOLO object detection Algorithm.

4.1 CycleGAN Experiments and Results

The CycleGAN is given two input datasets, namely the Real Image dataset and the Synthetic Image dataset. Below we can see how, iteration wise, a particular image gets transformed from one domain to another (Synthetic to Real) during the training period of CycleGAN :

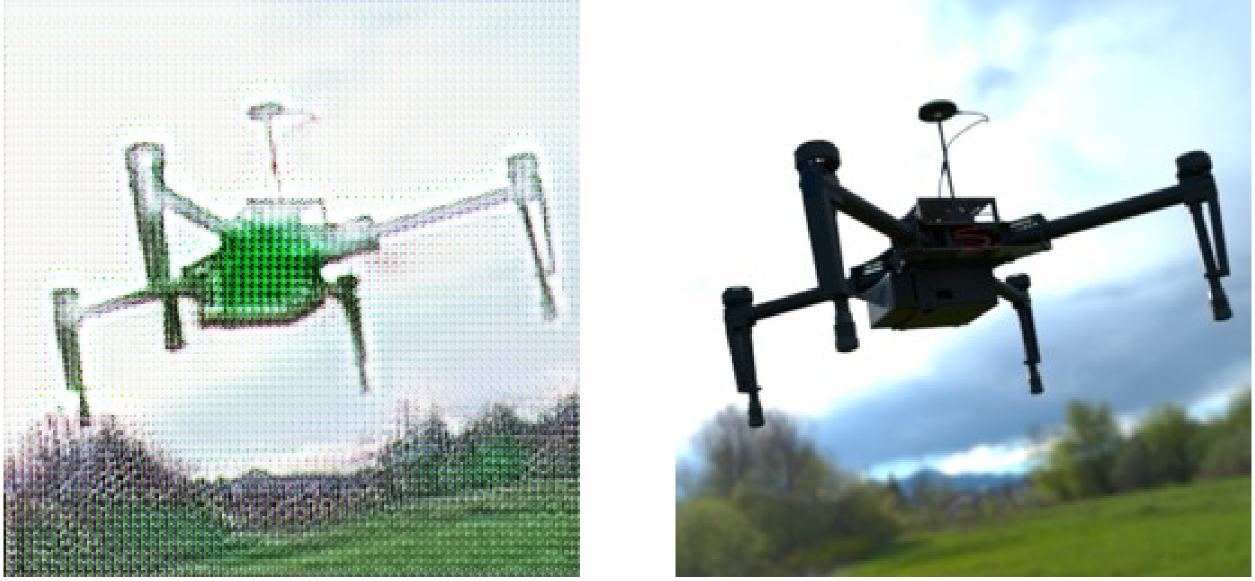


Figure 4.1: Results after 5 epochs

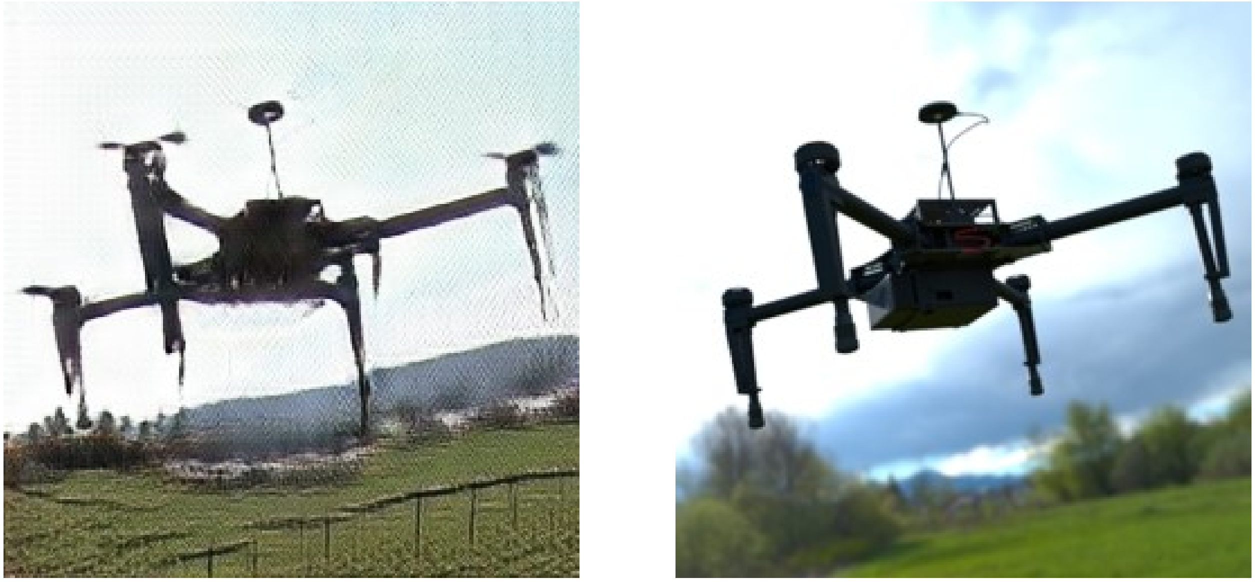


Figure 4.2: Results after 20 epochs

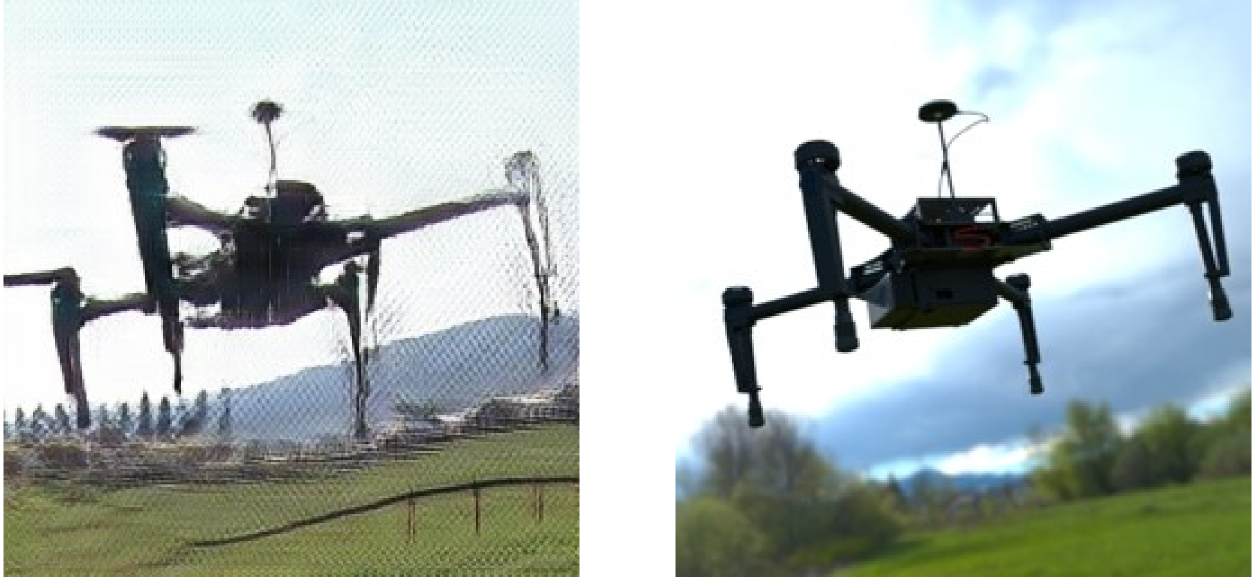


Figure 4.3: Results after 50 epochs

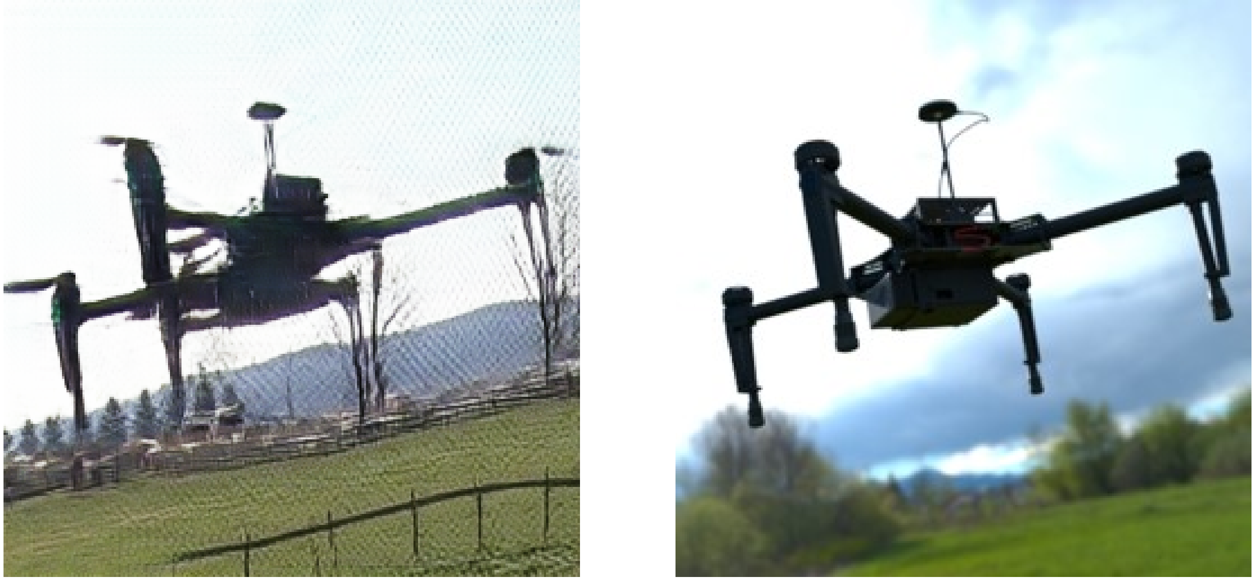


Figure 4.4: Results after 100 epochs

Some more generated images from CycleGAN from their respective Synthetic images can be shown below :

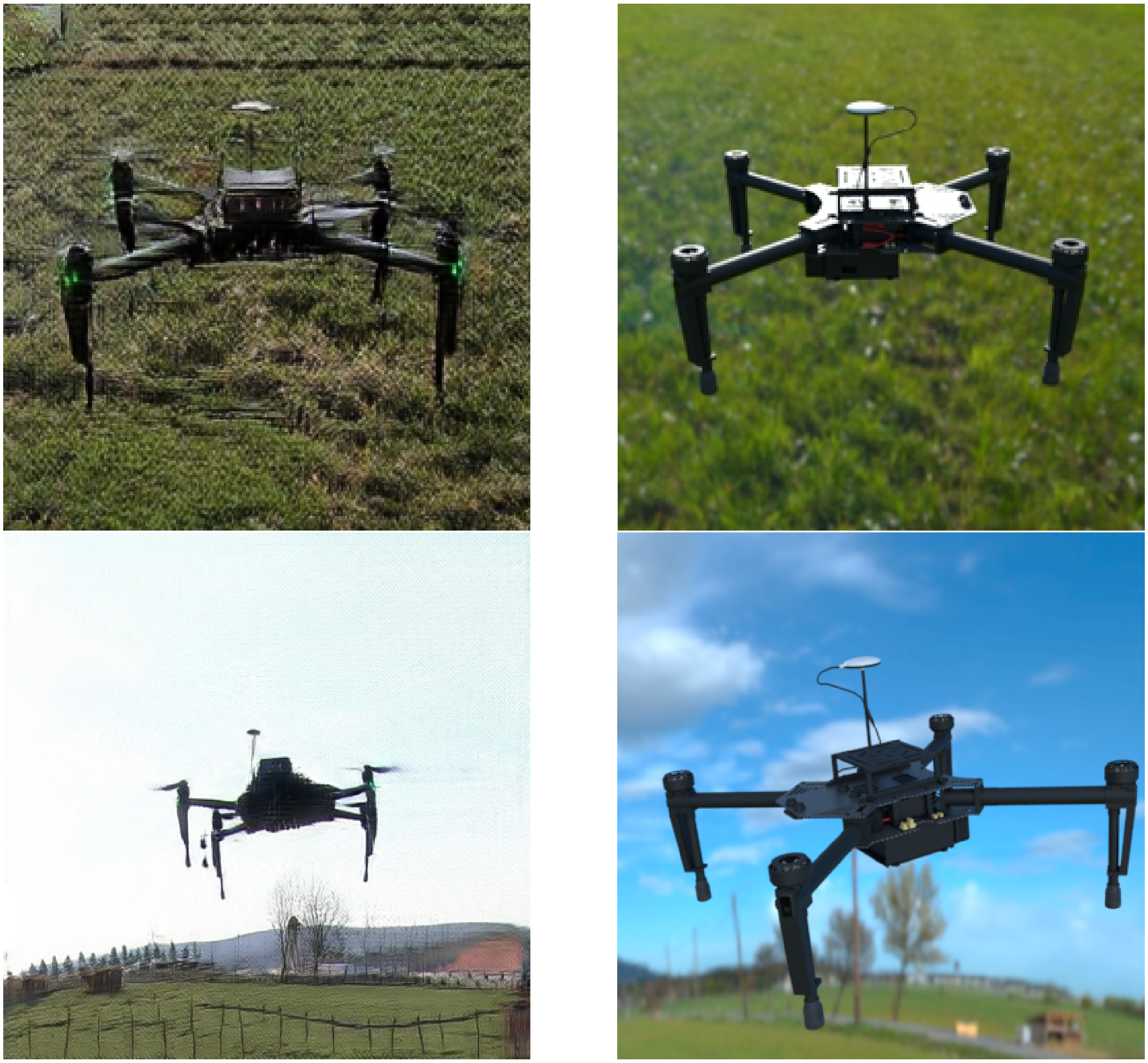


Figure 4.5: A few more generated images from CycleGAN are to the left, whereas their corresponding counter part synthetic images are to the right. In the above pictures, the transformation of images from one domain to another are seen clearly.

4.2 YOLO results and Experiments

The YOLO algorithm is trained and tested on different datasets so as to establish a fair way to evaluate the performance of each dataset. We have carried out the object detection model on four types of datasets. Each of their composition and results obtained are discussed below. In each of the cases, validation dataset contains images from its own dataset that it has not been trained on. The validation dataset consists of 10 percent of randomly chosen images from the dataset. The test dataset contains around 1500 images we see in the real world consisting of images that contain and around 500 images in the dataset do not contain images of drones and remain unlabelled.

4.2.1 Results on Synthetic Dataset

The synthetic dataset consists of images rendered from the 3D alone. Around 5,500 synthetic images were labelled for both the training and testing processes together. The model worked very well with the following values of accuracy :

- mAP : 99 percent

- IoU : 90 percent

Though it worked very well on the validation dataset, it could not perform well on the test dataset as it produced a mAP accuracy of 5.7 percent. The loss curve during it's training is plotted below :

Loss vs. Iterations Synthetic Dataset Training Curve.png Loss vs.bb

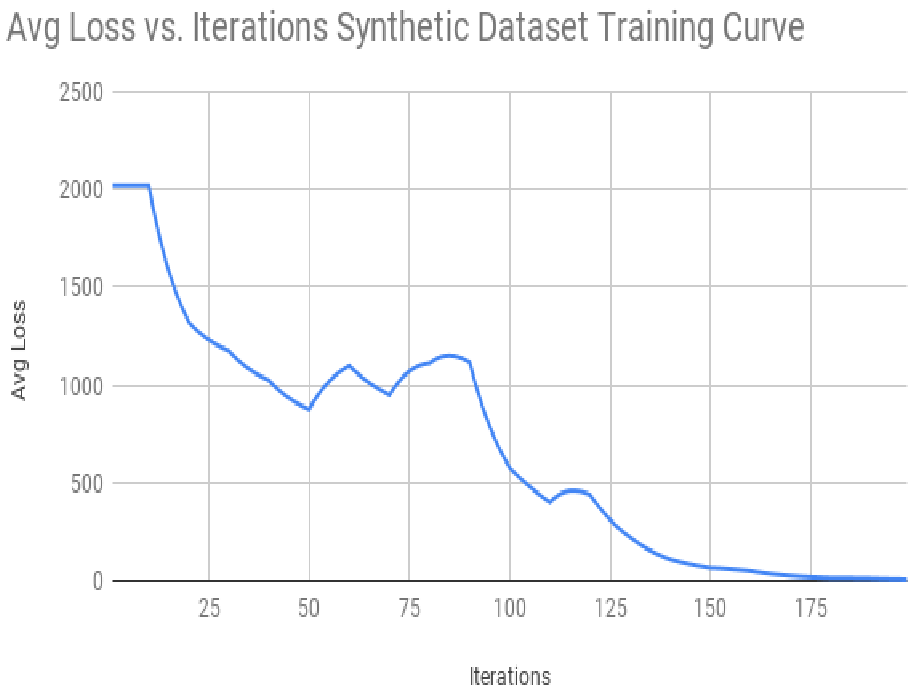


Figure 4.6: Loss Curve obtained during training process for Synthetic Dataset

4.2.2 Results on Real Dataset

The real images are obtained from the data collected though the videos which captured the drone in flight. For both the training and validation process, around 3000 images were used. The model worked well on both validation dataset and test dataset. The following are the performance results obtained for both the validation and test datasets respectively.

- mAP : 99 percent
- IoU : 81 percent

Accuracy values on the test dataset :

- mAP : 98.5 percent
- IoU : 64.76 percent

The loss curve obtained during the training process is shown below :

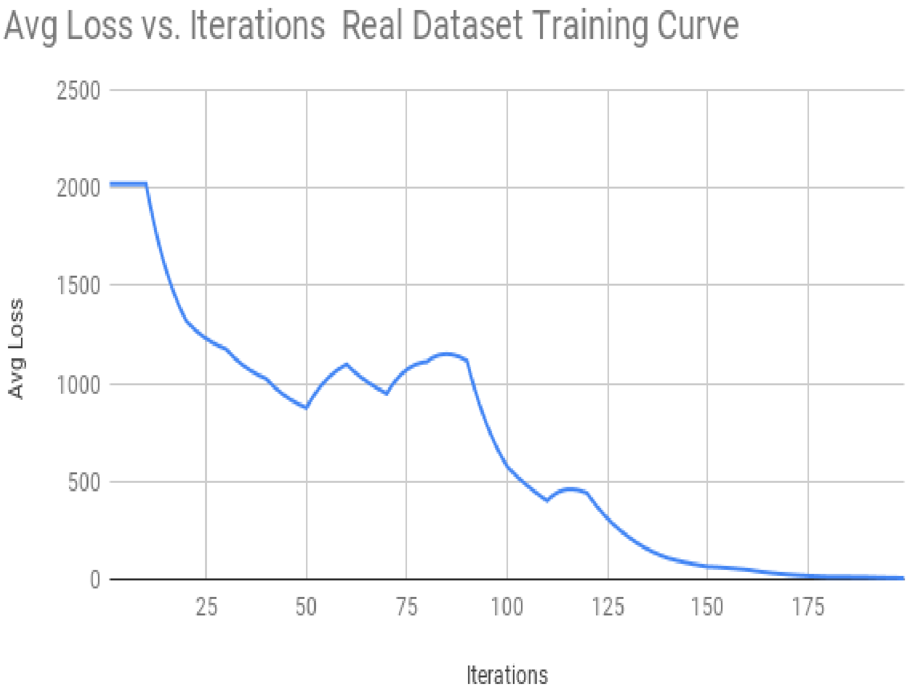


Figure 4.7: Loss Curve obtained during training process for Real Dataset

4.2.3 Results on Fake Dataset from CycleGAN

The images obtained that compose this dataset are the images that are generated from CycleGAN once the training process is complete. A set of 3000 synthetic images, which were not used to train CycleGAN, were sent to the the trained model where the results image distribution resembles that of the real image distribution. 3000 images were used for the process of training and validation. The following are the results obtained on the validation dataset:

- mAP : 82.5 percent
- IoU : 64 percent

Accuracy values when tested on the test dataset are :

- mAP : 91.54 percent
- IoU : 69 percent

The loss curve obtained during the training process is shown below :

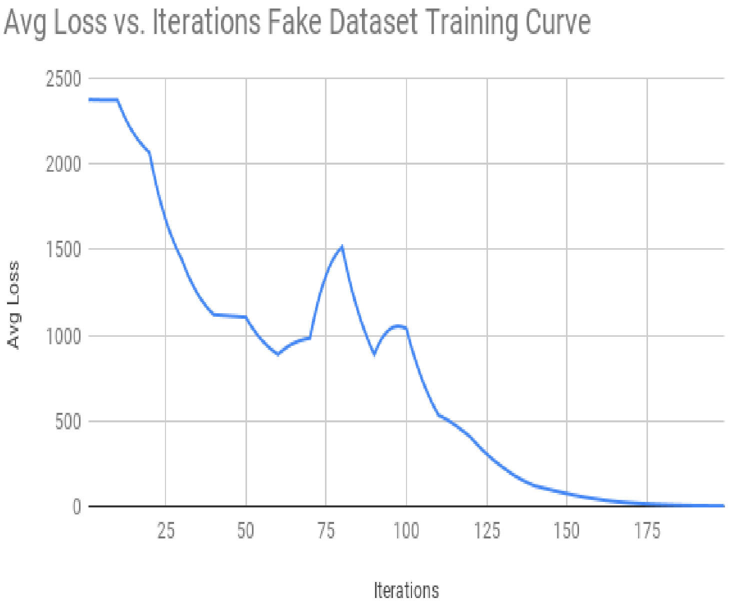


Figure 4.8: Loss Curve obtained during training process for Fake Dataset

4.2.4 Results on Mixed Dataset

This dataset consisted of 3000 labelled images each from the Real and Fake datasets. They were taken in the ratio 1:1 to see if there is any imporvement of accuracies over their respective datasets and the following accuracy values were observed. The following results are obtained when tested on the validation dataset :

- mAP : 97.5 percent
- IoU : 75 percent

Accuracy values when tested on the test dataset are :

- mAP : 99.2 percent
- IoU : 74 percent

The loss curve observed during the training is plotted below :

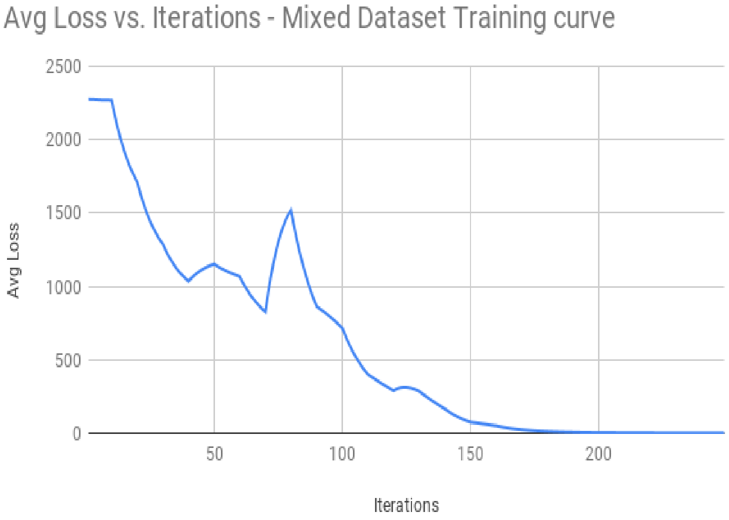
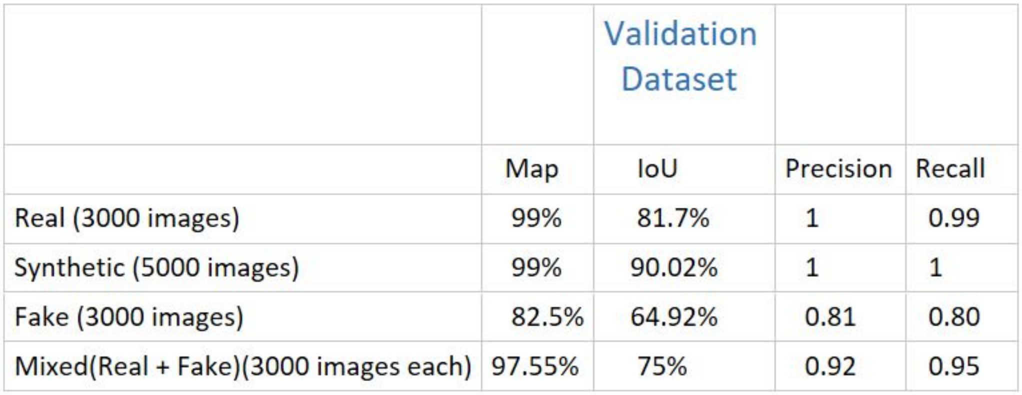


Figure 4.9: Loss Curve obtained during training process for Mixed Dataset

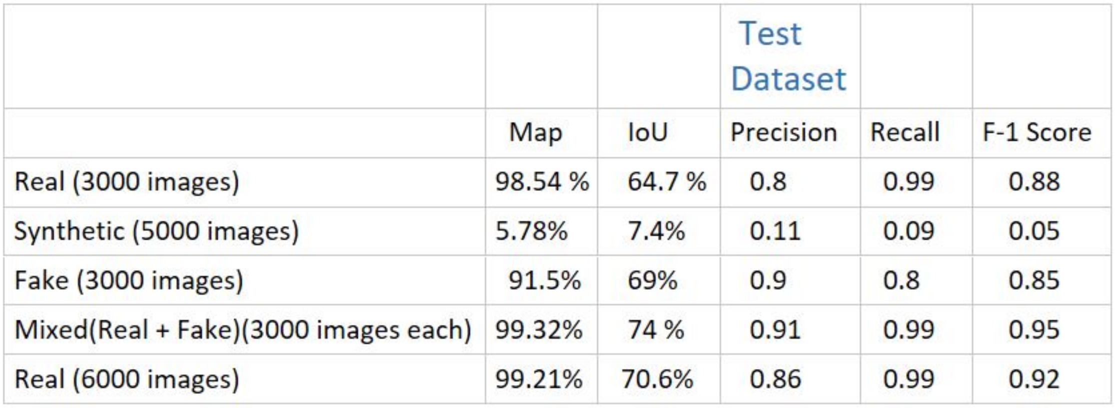
The table containing all the results of different datasets on Validation dataset is shown in the figure shown below 4.10.



		Validation Dataset		
	Map	IoU	Precision	Recall
Real (3000 images)	99%	81.7%	1	0.99
Synthetic (5000 images)	99%	90.02%	1	1
Fake (3000 images)	82.5%	64.92%	0.81	0.80
Mixed(Real + Fake)(3000 images each)	97.55%	75%	0.92	0.95

Figure 4.10: Results of different validation datasets on the Object Detction Model

Apart from the above mentioned datasets, the object detection model was also trained on a dataset consisting of 6000 real images as well. This is done so that there can be a proper comparison of results made between the Mixed dataset and the Real dataset, which are of the same size. The results of the performance of various datasets on the object detection model can be seen in the figure 4.11. In the figure 4.11, it can be seen how both the Mixed and the Real datasets consisting of 6000 images have very similar performance results. From this



			Test Dataset		
	Map	IoU	Precision	Recall	F-1 Score
Real (3000 images)	98.54 %	64.7 %	0.8	0.99	0.88
Synthetic (5000 images)	5.78%	7.4%	0.11	0.09	0.05
Fake (3000 images)	91.5%	69%	0.9	0.8	0.85
Mixed(Real + Fake)(3000 images each)	99.32%	74 %	0.91	0.99	0.95
Real (6000 images)	99.21%	70.6%	0.86	0.99	0.92

Figure 4.11: Performance of various models on the Test Dataset

result, we can make an observation on how real data could be supplemented by generated images from CycleGAN.

The Precision Recall curve for all the above mentioned datasets is plotted in the figure 4.12

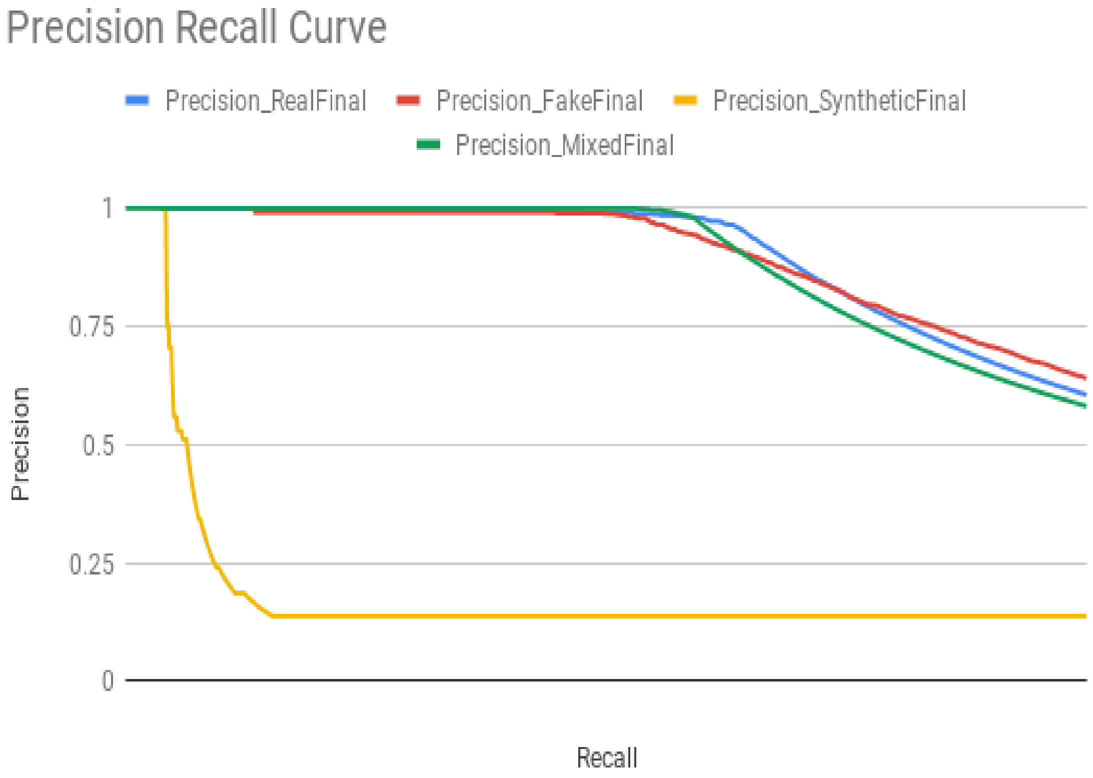


Figure 4.12: P-R Curves of all models on the Test Dataset. On Y-axis Precision values are plotted from 0 to 1 whereas on the X-axis, Recal values are plotted from 0 to 1

4.2.5 Final Detections with Bounding Box

Here are examples of few test images which were detected by the object detection model.

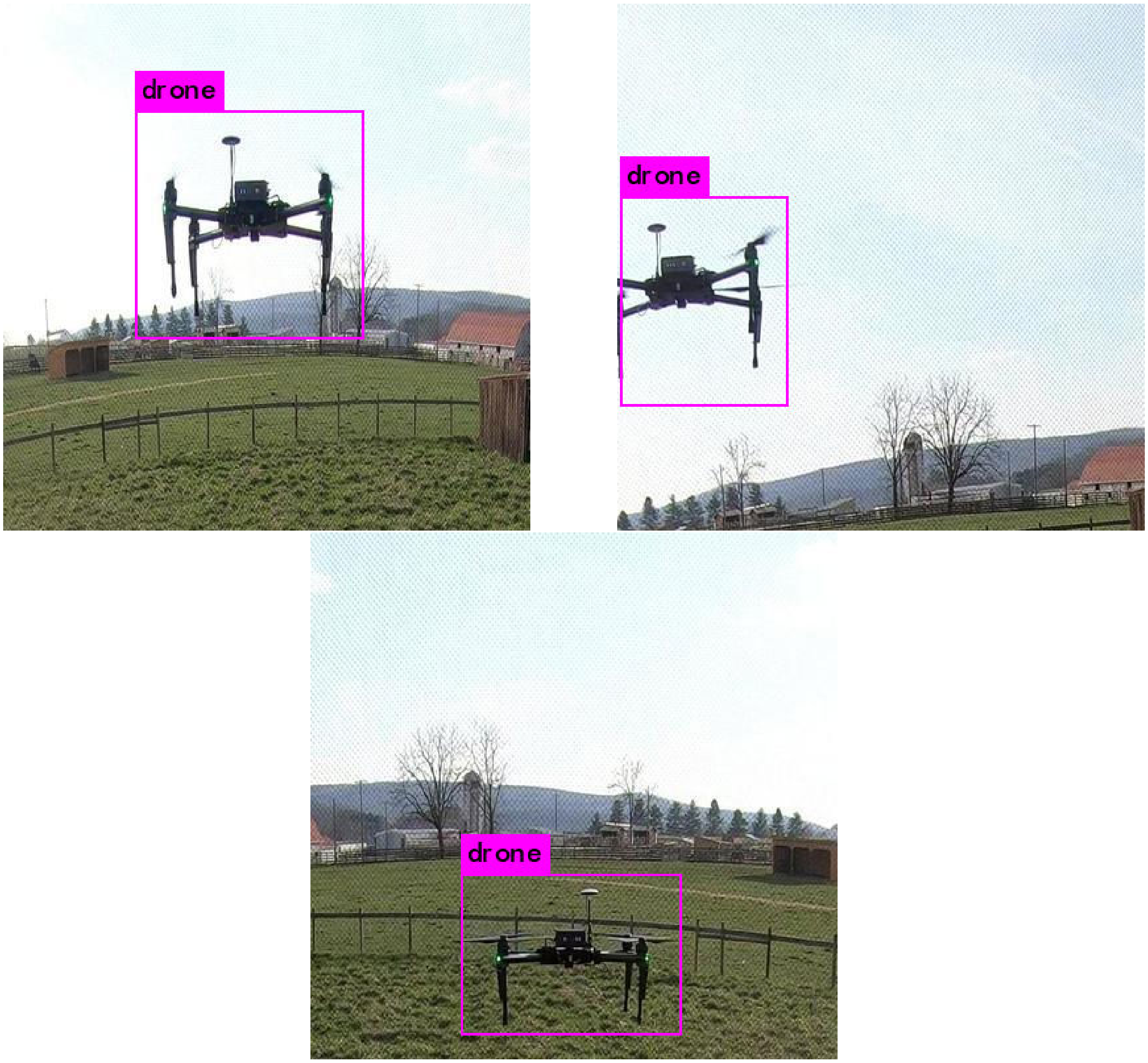


Figure 4.13: A few results from the object detection model

Chapter 5

Conclusions and Future Work

When we started out with this research, we aimed to see if Deep Learning techniques like Generative Adversarial Networks can help with our problem of data collection and generation of a huge amount of realistic data. CycleGAN[8] in theory seemed to align with what we were aiming to accomplish but we only got confident about this technique after looking at the realistic looking output images the trained model delivered. The CycleGAN model was successfully able to change the image distribution from the synthetic dataset to that of the domain of real images. Not only the background, even the resultant drone image seems to bare almost identical resemblance to the drone we used during our experiments without any problem.

The drone structure was maintained in most of the images and we were able to obtain images of drones in different angles and positions but with same image distribution as seen in the realistic images, mimicking it's environment, background etc. We knew that once we were able to train a model that could generate as realistic looking images from synthetic data, the next job in the pipeline was to add a downstream task to it to verify it's performance and see how viable this solution is. Suiting our needs, in order to see where the drone is in our field of view of another drone, we chose Object Detection as our downstream task, which is a well established problem in the field of Machine Learning and Deep Learning.

We chose one of the state of the art object detection model, YOLO(You Only Look Once), to find the performance of various datasets. All of the datasets were tested on 1500 real images. Though the model trained on real images performed very well on the test dataset, the model that trained on fake images alone, from CycleGAN, also performed quite well with an accuracy of 91.5 percent. Finally, it was observed that the model trained on Mixed dataset, containing both the real and generated fake images, was the best performing model. Through this way, it can be seen how there is a good possibility that the addition of fake images, generated from CycleGAN, has only helped the model in detecting the drones, but has not led to the decrease of accuracy by confusing the model.

As a part of the bigger picture, one of the main steps that has always been a part of the control pipeline was to measure the distance between two drones. Identifying the position of the drone in the field of vision of another drone has been one of the first steps to achieve that. Now that the drone can be localized, the next step would be to come out with ways to measure how far the target drone is from the other respective drone. Though the training and testing datasets involved images with no drones, there is a possibility that the model would learn better if it was trained on other classes of objects which bared resemble to the structure of the drone, like a bird or an airplane etc, which are some of the common object we might find in the sky as well. One of the other important plans for the future works is to train the CycleGAN using different 3D models of drones, which are available in our lab. After the CycleGAN, few other GAN models were built as an extension to CycleGAN where they try to improve translation learning across large geometry variations[49] and enforce a better consistent mappings during the translation[50]. We can try using these advanced GANs in place of CycleGAN to see if there is an performance improvement in the generated images obtained over the prior method.

5.1 Observations

- The CycleGAN model is successfully able to change the image distribution from the synthetic dataset to that of the domain of real images. Not just the background of the image, even the drone in the generated image seems to bare alot of resemblance to the drone we used during our experiments without any problem.

- It has been observed that increasing the diversity within images and the number of images used, gives better results during the training process of CycleGAN with much less number of iterations.

- Roughly about 20 percent of the synthetic images, used in the testing phase for image translation, turned out to have a distorted output. This could be due to the possibility that there was a great difference in the background between the images seen in those synthetic dataset when compared to the background we observe in the real images.

- Another interesting observation made during the evaluation of CycleGAN is that, though the initial drone angle and position has been maintained for a lot of images, this cannot be quite clearly seen in the case of a few images though the structure of the drone seems to be preserved.

- The mixed dataset evaluation results on the test dataset were very similar to that of the real dataset results when compared against each other. This shows how the addition of data is only helping the model detect the drones, but has not decreased the accuracy by confusing the model.

- When detection model was trained on 6000 real images, the performance for those number of real images versus the mixed dataset was essentially identical which indicates that we can use fake data to supplement real data.

Bibliography

[1] "Everything you need to know about neural networks." https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491. Accessed:2019-06-08.

[2] R. Elbers and T. Heskes, "On the replication of cyclegan," 2018.

[3] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[4] "Different kinds of convolutional filters." https://www.saama.com/blog/different-kinds-convolutional-filters/. Accessed:2019-06-08.

[5] "A comprehensive guide to convolutional neural networks—the eli5way." https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53/. Accessed:2019-06-08.

[6] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui, and J. Collomosse, "Everything you wanted to know about deep learning for computer vision but were afraid to ask," in *2017 30th SIBGRAPI conference on graphics, patterns and images tutorials (SIBGRAPI-T)*, pp. 17–41, IEEE, 2017.

[7] "Artificial data generation with generative adversarial networks (gans) – part 2: Cyclegan." https://neurosys.com/article/artificial-data-generation-with-generative-adversarial-networks-gans-part-2-cyclega Accessed:2019-06-08.

[8] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[9] "Understanding and implementing cyclegan in tensorflow." https://hardikbansal.github.io/CycleGANBlog. Accessed:2019-06-08.

[10] "Introduction to cyclegan." https://rubikscode.net/2019/02/04/introduction-to-cyclegan/. Accessed:2019-06-08.

[11] "How to implement a yolo (v3) object detector from scratch in pytorch: Part 1." https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/. Accessed:2019-06-08.

[12] "What is object detection? introduction to yolo algorithm." https://appsilon.com/object-detection-yolo-algorithm. Accessed:2019-06-08.

[13] "Non-max suppression." https://www.coursera.org/learn/convolutional-neural-networks/lecture/dvrjH/non-max-suppression. Accessed:2019-06-08.

[14] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[15] B. A. S. S. Redmon, J., "Darknet: Yolov3 - neural network for object detection." https://github.com/AlexeyAB/darknet, 2019.

[16] "Intersection over union (iou) for object detection." https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. Accessed:2019-06-08.

[17] “map (mean average precision) for object detection.” https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173/. Accessed:2019-06-08.

[18] J. Wang and E. Olson, “Apriltag 2: Efficient and robust fiducial detection,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198, IEEE, 2016.

[19] Y. Chen, P. Aggarwal, J. Choi, and C.-C. Jay, “A deep learning approach to drone monitoring,” in *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pp. 686–691, IEEE, 2017.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[22] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

[23] D. F. Specht, “A general regression neural network,” *IEEE transactions on neural networks*, vol. 2, no. 6, pp. 568–576, 1991.

[24] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, “Learning activation functions to improve deep neural networks,” *arXiv preprint arXiv:1412.6830*, 2014.

[25] J. Shore and R. Johnson, "Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy," *IEEE Transactions on information theory*, vol. 26, no. 1, pp. 26–37, 1980.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[28] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*, pp. 92–101, Springer, 2010.

[29] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[30] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[31] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in neural information processing systems*, pp. 2234–2242, 2016.

[32] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[33] M.-Y. Liu and O. Tuzel, "Coupled generative adversarial networks," in *Advances in neural information processing systems*, pp. 469–477, 2016.

[34] Y. Aytar, L. Castrejon, C. Vondrick, H. Pirsiavash, and A. Torralba, "Cross-modal scene networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 10, pp. 2303–2314, 2017.

[35] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2107–2116, 2017.

[36] Y. Taigman, A. Polyak, and L. Wolf, "Unsupervised cross-domain image generation," *arXiv preprint arXiv:1611.02200*, 2016.

[37] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3722–3731, 2017.

[38] J. Liu and R. K. Williams, "Optimal intermittent deployment and sensor selection for environmental sensing with multi-robot teams," in *IEEE International Conference on Robotics and Automation*, pp. 1078–1083, IEEE, 2018.

[39] M. Wu, W. Xie, X. Shi, P. Shao, and Z. Shi, "Real-time drone detection using deep learning approach," in *International Conference on Machine Learning and Intelligent Communications*, pp. 22–32, Springer, 2018.

[40] M. Saqib, S. D. Khan, N. Sharma, and M. Blumenstein, "A study on detecting drones using deep convolutional neural networks," in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–5, IEEE, 2017.

[41] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and

super-resolution," in *European conference on computer vision*, pp. 694–711, Springer, 2016.

[42] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2414–2423, 2016.

[43] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images.," in *ICML*, vol. 1, p. 4, 2016.

[44] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networkss," in *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[46] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

[47] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[48] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[49] W. Wu, K. Cao, C. Li, C. Qian, and C. C. Loy, “Transgaga: Geometry-aware unsupervised image-to-image translation,” *arXiv preprint arXiv:1904.09571*, 2019.

[50] R. Zhang, T. Pfister, and J. Li, “Harmonic unpaired image-to-image translation,” *CoRR*, vol. abs/1902.09727, 2019.