3D OBJECT DETECTION USING VIRTUAL ENVIRONMENT ASSISTED DEEP NETWORK TRAINING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Ashley S. Dale

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

December 2020

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Sola Dei gratia

ACKNOWLEDGMENTS

Deepest thanks to my advisor Dr. Christopher for her continued mentorship, guidance, advice, encouragement, restraint, and letting me play with all the lab GPUs. Words fail to adequately describe my affection for her and the profound gratitude I have for her impact on my life. Thank you.

A huge thanks to the team of Media Arts and Sciences (MAS) researchers at IUPUI led by Albert William and managed by Wen Krogg. They were responsible for creating the synthetic data sourced for this research, with extensive collaboration between the MAS and engineering teams to define the synthetic data set characteristics and parameters. Additional shout-outs to Emma Overmeyer, Aika Callahan, Sarah Dale, and Susan Dale for their contributions to data creation, collection, and processing.

Many thanks to Dr. David Emerson for his excellent instruction in good coding and debugging practices, sharing cool analytical tools, answering a huge number of questions, and being a brain-storming buddy.

Additional thanks to Rob Meagher and the CNC Help Desk team for keeping the computers running, and to T. Bsaibes for tea and sympathy.

Recognition is definitely due to administrative staff Sherrie Tucker and Jennifer Watson for having my back and keeping things on track. Sherrie's "fairy-dust" is by far unmatched.

I gratefully acknowledge the Office of Naval Research Grant Number N00014-20-1-2674 and SAIC subcontract P010238559 for financial support during this research.

Finally, I need to acknowledge the constant love, support and encouragement of my grandparents, parents, and sisters. You have all contributed in uncountable ways, both big and small, to this work; I am so grateful for each of you. My love to you all forever.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABBREVIATIONS

AI	artificial intelligence
CNN	convolutional neural net
DNN	deep neural net
FPN	feature pyramid network
mAP	mean Average Precision
ML	machine learning
MR-CNN	Mask R-CNN
MS COCO	Microsoft Common Objects in Context (data set)
R-CNN	Regions with CNN Features (Network)
RoI	Region of Interest
RW	real world
VAE	variational autoencoder
VW	virtual world

NOMENCLATURE

RGB	Real-world RGB image data
RGB*	Virtual-environment RGB image data
RGBZ*	Virtual-environment image data containing depth information
RGBZ	Real-world RGB image data with manually-created artificial depth map
RGB(Z*)	The comprehensive set of all RGB, RGB*, RGB(Z), and RGBZ* data
VW-RGB*	The subset of RGB* data consisting of objects placed in virtual environments
360-RGB*	The subset of RGB* data consisting of 360-degree rotations

ABSTRACT

Dale, Ashley S. MSECE, Purdue University, December 2020. 3D Object Detection Using Virtual Environment Assisted Deep Network Training. Major Professor: Lauren Christopher.

An RGBZ synthetic dataset consisting of five object classes in a variety of virtual environments and orientations was combined with a small sample of real-world image data and used to train the Mask R-CNN (MR-CNN) architecture in a variety of configurations. When the MR-CNN architecture was initialized with MS COCO weights and the heads were trained with a mix of synthetic data and real world data, F1 scores improved in four of the five classes: The average maximum F1-score of all classes and all epochs for the networks trained with synthetic data is $F_1^* = 0.91$, compared to $F_1 = 0.89$ for the networks trained exclusively with real data, and the standard deviation of the maximum mean F1-score for synthetically trained networks is $\sigma_{F1}^* = 0.015$, compared to $\sigma_{F1} = 0.020$ for the networks trained exclusively with real data. Various backgrounds in synthetic data were shown to have negligible impact on F1 scores, opening the door to abstract backgrounds and minimizing the need for intensive synthetic data fabrication. When the MR-CNN architecture was initialized with MS COCO weights and depth data was included in the training data, the network was shown to rely heavily on the initial convolutional input to feed features into the network, the image depth channel was shown to influence mask generation, and the image color channels were shown to influence object classification. A set of latent variables for a subset of the synthetic datatset was generated with a Variational Autoencoder then analyzed using Principle Component Analysis and Uniform Manifold Projection and Approximation (UMAP). The UMAP analysis showed no meaningful distinction between real-world and synthetic data, and a small bias towards clustering based on image background.

1. INTRODUCTION

The usefulness of a machine learning approach to potential computer vision tasks has been widely demonstrated, including computer vision tasks such as traffic analysis (with implications for autonomous vehicles) [1–3], quality control during manufacturing (with implications for task automation) [4–6], and face recognition [7–9]. Independently, these applications represent another incremental improvement in performance for each task; collectively they present evidence of a machine learning revolution, where artificial intelligence is developing the capability to compete with human vision.

In the normative model of vision, a pair of eyes provide visual data (comprised of color and depth information) to the brain for analysis [10]. Keeping pace with the improvement in hardware, computer vision data has transitioned from grayscale [11] to color [12] and, with the introduction of the Microsoft Kinect in 2010, now frequently includes depth data [13–15].

Historically, image depth data has been captured through the use of paired stereoscopic images, essentially reproducing the view from each eye in the vision model, and much effort has been invested in accurately inferring depth from stereoscopic images [16–20]. However, the introduction of the Microsoft Kinect opened the door to a relatively low cost, widely available method of mass gathering depth data directly, without an additional depth-inference step. The algorithm is now freed from the burden of inferring depth and may instead dedicate all computational resources to the task of processing the additional information.

In fact, the issue of limited resources (i.e. time, data, computational power) continues to hinder progress in accomplishing computer vision tasks, with significant literature dedicated to the subject of resource optimization by decreasing training time and required computational power and implementing data augmentation [21–29].

Recent progress in transfer learning shows that it may be possible to comprehensively train a single network once, then tune the generalized network for a specific task by relaxing the weights on specific layers [30,31], reducing the long term training burden and reducing the required computational resources needed during the training process. Similar advances in virtual environment creation alleviate the need to obtain large data sets from real-world environments subject to unpredictable weather, lighting, and other factors [32–34]. Reducing the overhead cost of machine learning computer vision solutions by using these approaches allows potential solutions to be more widely implemented.

However, throughout all of these resource optimization approaches the final goal remains the same: the reproduction of vision capabilities in real-world environments. To that end, it is worth testing if transfer-learning and synthetic data approaches may be combined, the effectiveness of combining the approaches, and the improvement in overall performance over previous methods.

The task chosen is the detection and subsequent classification of the following objects: plane, hang glider, kite, quadcopter, and eagle. Object classification refers to the labeling of a pre-localized object in an image, whereas object detection refers to the localization of an object within an image followed by the correct classification.

Two data sets were created using these objects: The first data set consists of real-world images gathered from the internet, and is denoted as the RGB data set. The second data set consists of virtual objects modeled after the real-world but placed in an entirely synthetic, virtual world, and is denoted RGB*. Since the virtual environment allows for complete control over the data gathering process, a subset of the RGB* data set containing virtual z-depth maps was created; the synthetic data set containing these depth maps is denoted RGBZ*. The RGB data did not include depth information, so to enable comparison between RGB data and the RGBZ* data depth maps were created for each RGB image by hand: This RGB data set expanded with the artificial depth maps is termed RGBZ. The RGB and RGBZ* data sets were used to study the impact of synthetic data from virtual environments on the performance of a machine learning algorithm for object detection.

In the following sections of this chapter, an overview of the state of the art on object detection machine learning algorithms and the metrics used to evaluate network performance are presented to provide context for this work. In subsequent chapters, the effects of including synthetic data with real world data and the effects of including depth data during training are presented with the necessary details, and an attempt at understanding how these results arise from data set features is presented.

1.1 Machine Learning Algorithms for Object Detection

As recently as 2015, every state-of-the-art machine learning algorithm relied solely on 2D image data [35]. Challenges such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) popularized the use of Deep Neural Networks (DNNs) for object detection tasks, where many of the DNN architectures implemented convolutional layers [36] and are therefore termed Convolutional Neural Networks (CNNs).

Although many fascinating and capable CNN architectures and algorithms have been developed in the past decade, the focus for the following sections is narrowed to the R-CNN family of architectures and the VGG-ResNet CNNs culminating in the introduction of Mask R-CNN in 2017. Since the Mask R-CNN architecture is a combination of the R-CNN and ResNet architectures, brief histories and descriptions of the R-CNN and ResNet architectures are presented since the Mask R-CNN architecture is a combination of both architectures, while the Mask R-CNN architecture is presented in detail in section 1.1.3.

1.1.1 The R-CNN Network Architectures

In 2014, Girshick et al. focused attention on feature-based object classification through the introduction of a machine learning network architecture consisting of a region proposal architecture with a backbone of convolutional layers named “Regions with CNN features” or R-CNN [37].

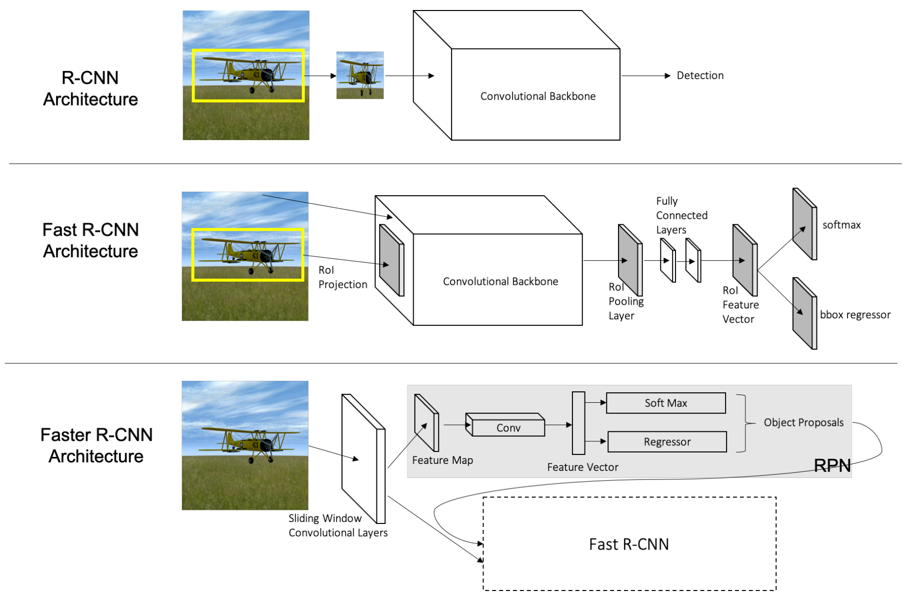


Fig. 1.1.: The architectures for R-CNN [37], Fast R-CNN [38], and Faster R-CNN [39]. Each architecture gains capability as complexity increases; while R-CNN basically consists of only a CNN and modified input, Fast R-CNN comprises several sub-networks, and Faster R-CNN includes the Region Proposal Network (labeled RPN and shaded gray) and the entire Fast R-CNN architecture.

Before R-CNN, the most successful image classification and object detection algorithms relied on finding various pixels of interest within an input image, then searching surrounding pixels to create a set of features associated with the detected object [11, 12, 40]. These algorithms achieved a mean Average Precision (mAP) of approximately 30% on standard classification tasks; in comparison, through the use of convolutional layers, R-CNN achieved an astounding 53.7% [37].

Girshick et al. drew inspiration from the visual system to combine this region-based approach with the insight that—-in the visual system—-object detection and classification occur at several different levels along the vision pathway. Accordingly, R-CNN divides the object detection and classification task into two components: localization by generating proposal regions-of-interest (RoIs), and feature detection through the use of CNNs. To tune R-CNN to a specific task (transfer learning), the classification layer is replaced and the rest of the architecture is left unaltered [37].

R-CNN was supplanted in 2015 when Girshick introduced Fast R-CNN [38]. Training Fast R-CNN is at minimum 9x faster than R-CNN, an achievement accomplished by computing a convolutional feature map once, then sharing the map with all future computations (thereby reducing the computational burden) [38]. While R-CNN innovated the region proposal with the convolutional layer, Fast R-CNN introduced multiple innovations such as multi-task loss and updating all layers during training (specifically convolutional layers), eliminating the need for writing features to disk [38]. The Fast R-CNN paper also describes the protocol for adapting the network for transfer learning with pre-trained networks: the max pooling layer is replaced with an RoI pooling layer compatible with the pre-trained network, the last fully connected layer and softmax layer are replaced with layers of the same kind but specific to the new task, and the network input is modified to accept a list of images and a list of the image RoIs [38].

One further improvement was made to the Fast R-CNN network architecture in 2015 with the introduction of the Region Proposal Network (RPN), effectively eliminating the region proposal computational burden, allowing detections in real-time, and giving the world Faster R-CNN [39]. The RPN accepts any size image as input to a convolutional layer, uses a small network as a sliding window over the convolutional layer feature map, and maps the sliding window to a lower dimensional vector which is then fed into the CNN backbone [39].

The architectures for R-CNN, Fast R-CNN, and Faster R-CNN are shown in Figure 1.1. These architectures addressed many of the major issues (speed, memory, data, accuracy, etc.) hindering machine learning object detection and classification algorithms with innovations now widely implemented in other network architectures. These issues were addressed by placing the CNN architectures (typically VGG-16) into a new computational setting, adopting a modular approach that combined the best CNN for the job with additional sub-networks specific to the task at hand [37,37,39].

However, accuracy was not yet 100%, and the R-CNN architectures by definition relied on CNN architectures which were still struggling with their own set of training issues and challenges.

1.1.2 ResNet

Thanks to successes at challenges such as ILSVRC, multi-layer CNNs continued to grow deeper and deeper, which introduced the question *how deep is too deep?* [41].

The Visual Geometry Group (VGG) at the University of Oxford designed two CNNs specifically to answer this question [41]. The first architecture known as VGG-16 included sixteen convolutional layers, while the second architecture VGG-19 included nineteen convolutional layers [41]. However, machine learning practitioners discovered that as the networks grew deeper accuracy plateaued then quickly degraded [42].

A solution for the accuracy degradation was found in 2016, when He et al. introduced ResNet, a variation on the VGG architectures [42]. VGG architectures implement a sequential flow through each network layer; ResNet created shortcuts, allowing inputs to slip past groups of layers and therefore "survive" to affect the training of deeper network layers [42]. The improvement with the ResNet architecture was drastic, allowing a 1001 layer network to be trained [42,43] and winning first place at the ILSVRC 2015 Classification task with a 152 layer CNN [42].

In total, He et al. introduced four ResNet architectures: ResNet-34, ResNet-50, ResNet-101, and ResNet-152 with 34, 50, 101, and 152 layers respectively [42]. ResNet-34 was the least accurate of the four architectures, and in general, deeper networks once again produced higher quality results [42]. Unsurprisingly, there was a shift away from using VGG-style architectures as the convolutional backbones for network architectures, including a reworking [44].

1.1.3 Mask R-CNN

By 2017, the Faster R-CNN architecture with various CNN backbones was state-of-the-art for object detection and classification tasks. However, a comparison of Faster R-CNN with VGG-16, ResNet-50, googLeNet, and caffeNet backbones highlights that, while Faster R-CNN/ResNet-50 had the highest accuracy, it also had the slowest computational time [44], and therefore fell short of truly real-time image analysis. Faster R-CNN was also incapable of instance segmentation tasks, specifically the pixel-by-pixel detection and labeling of an occluded object [45]. To address these issues, He et al. added a new sub-network to the Faster R-CNN architecture specifically for the task of predicting a segmentation mask on each RoI (following the modular approach begun with Fast R-CNN and continued with Faster R-CNN) and dubbed the modified architecture Mask R-CNN (MR-CNN) [45]. The new sub-network runs in parallel with the network heads, as shown in Figure 1.2.

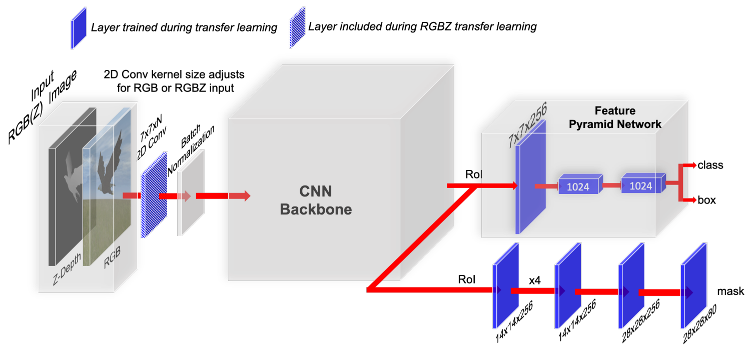


Fig. 1.2.: Mask R-CNN Architecture with Feature Pyramid Network (FPN) variant [45]. For this work, ResNet-50 and ResNet-101 CNN backbones were implemented. An example RGBZ image is shown for illustrative purposes; training was conducted with independent RGB(*) and RGBZ* data sets. Transfer learning with Mask R-CNN required excluding specific layers from the pre-trained weights (shown in blue). When transfer learning with depth data, an additional layer was excluded from the pre-trained weights; this layer is shown as blue stripes.

The MR-CNN paper evaluates several different backbone/head configurations by combining the ResNet and ResNeXt -50 and -101 CNNs with an additional convolutional layer used to extract features from the final convlutional layer of the fourth-stage or with a Feature Pyramid Network (FPN) [45]. The FPN helps keep scale invariance by taking advantage of the fact that the CNN backbone already extracts features at various scales, essentially re-creating a feature pyramid as a byproduct of the main computational stream by sampling the convolutional layers at various points. This provides a significant improvement in overall performance at low computational cost [45, 46].

As of writing, MR-CNN continues to be state-of-the-art for machine learning tasks requiring object detection from images. Accordingly for this work, MR-CNN with a ResNet-(50,101) backbone and the FPN option was implemented from the Matterport GitHub repository [47].

Transfer learning with RGB* data is discussed in detail in Chapter 2, followed by a discussion of transfer learning with RGBZ* data in Chapter 3 and analysis of the data set feature space in Chapter 4.

1.2 Evaluation Metrics

When evaluating input data, the Mask R-CNN network returns detections with a mask, a box extracted from the mask, and a confidence score per class for each detected instance. The detected instance is assigned to the class with the highest confidence score.

Additional evaluation is required to determine if the returned results are valid. False detections, poorly fitting masks/boxes, and incorrect class assignments must all be identified to quantify network performance.

Two metrics were used to evaluate network performance: the Frame Detection Accuracy (FDA) metric and the F1-score. The FDA metric was used to evaluate each detection as either a true-positive detect or a false-positive detect based on various user-defined input parameters. This true/false designation was then used to calculate the F1-score for each class and the data set in its entirety. Together, the FDA and F1-score quantify the Mask R-CNN performance on the entire RGB(Z*) data set.

1.2.1 Frame Detection Accuracy Metric

The Video Analysis and Content Extraction (VACE) metrics introduced two distinct measures to evaluate detection and tracking [48]. The Frame Detection Accuracy (FDA) metric evaluates the performance of the network detection on a single frame, while the Sequence Frame Detection Accuracy (SFDA) considers how the network performed across a sequence [48]. For this study, the FDA metric was implemented to evaluate the quality of the results returned by the classifier.

To implement the FDA metric, first the overlap ratio per image is determined by summing the Intersection over the Union (IoU) of detected objects with ground truth objects using the relation

$$Overlap\ Ratio = \sum_{i=1}^{N_{mapped}^{(t)}} \frac{\left|G_i^{(t)} \cap D_i^{(t)}\right|}{\left|G_i^{(t)} \cup D_i^{(t)}\right|} \tag{1.1}$$

where, for input image t, $G_i^{(t)}$ is the ith ground-truth object and $D_i^{(t)}$ is the ith detected object.

The VACE overlap ratio allows for omitting detections with IoU values below a given threshold. If a detection has an IoU below the threshold, it is considered to be a false-positive detection and does not contribute to the overlap ratio. Detections with an IoU above the threshold are considered a true-positive detection. The overlap ratio is then used to determine the accuracy of detections per image using the VACE FDA metric defined as

$$FDA = FrameDetectionAccuracy = \frac{Overlap\ Ratio}{\frac{1}{2}(N_G^{(t)} + N_D^{(t)})} \tag{1.2}$$

where $N_G^{(t)}$ is the number of ground truth objects and $N_D^{(t)}$ is the number of detections in image t.

In this implementation, the class assigned to the detection must match the class of the ground truth for the IoU to return a true-positive value. The number of missed detections (false-negatives) was simultaneously determined with the FDA by comparing the true-positives to the ground-truth. The true-positives, false-positives, and false-negatives were then used to determine the F1-score. A lenient IoU threshold of 30% was set to emphasize class labeling accuracy in the reported statistics to allow study of where the network architecture failed to return a highly tuned mask and bounding box.

1.2.2 F-Score

The F-score is used in conjunction with the VACE detection metric to evaluate the network's ability to classify detections correctly. Also known as the "F-metric" or "F1-score" when $\beta = 1$, this measure is ubiquitous within machine learning algorithm evaluation as a single-number measure of network performance. Mathematically, the F1-score is the harmonic mean of precision and recall where

$$P = precision = \frac{N_{correct}}{N_{correct} + N_{incorrect}} \tag{1.3}$$

$$R = recall = \frac{N_{correct}}{N_{possible}} \tag{1.4}$$

and

$$F_\beta = \frac{(\beta + 1)PR}{\beta^2 P + R} \tag{1.5}$$

where $\beta = 1$.

Although multi-class versions of the F1-score exist, in this study the F1-score considers each class independently. When calculating precision and recall, the values for $N_{correct}$ and $N_{possible}$ are taken from the ground truth within a particular class.

However, $N_{incorrect}$ includes all relevant misclassifications from every class. E.g., for two classes [bird = 10 instances, plane = 10 instances], if 3 instances of bird are mislabeled as plane, and 9 instances of plane are mislabeled as bird, when the F1-score for the plane class is determined, $N_{incorrect} = 3$, $N_{correct} = 1$, and $N_{total} = 10$. As a result, the precision contains all the information from every class that is mapped to a single class whether erroneous or not, while the recall is class specific. This method of calculating the F1-score allows for evaluation of the interplay between classes during training.

The F1-score has been criticized for weighting precision and recall equally [49, 50] and for its inability to distinguish between classes [50, 51]. In this study, evaluation of the network's ability to recall features determined from synthetic data and/or depth data is of equal relevance to the network's ability to correctly classify an instance by applying learned features. Therefore, an equal weighting of precision and recall is a benefit of the F1-score, not a detriment.

2. REAL WORLD AND VIRTUAL WORLD: INCLUSION OF SYNTHETIC DATA DURING TRAINING

2.1 Introduction

The role of synthetic data in augmenting real-world data to improve machine learning results is of great interest to artificial intelligence researchers. With applications ranging from determining local order in spatial regimes [52] to aiding in the prevention of money laundering [53] or phenotyping tumors [54], combining synthetic data with real-world data has been shown to introduce diversity into a data set to prevent overfitting [55] and reduce the manual labor required to create and annotate data sets [32].

We show that it is possible to achieve outstanding machine learning results by combining a very small set of real-world data with a large set of synthetic data when using the Mask-RCNN architecture and transfer learning on the Microsoft Common Objects in Context (MS COCO) data set.

2.2 Generation of Synthetic Data

Generation of the synthetic data set was completed by a team of Media Arts and Science (MAS) students under the guidance of Albert William and Wen Krogg. The MAS team used AutoDesk Maya, a fully featured modeling software capable of simulating objects and environments with intricate textures, lighting, and physical motion. This software was used to model various real-world objects, then render images of these objects in two different approaches to generating synthetic data.

The first approach used by the MAS team was to create fully developed virtual environments that mimic natural or urban landscapes. Houses, trees, cars, etc. were modeled in Maya, imported into a virtual Maya environment, then animated to produce a series of images with high verisimilitude to images sourced from the real world. Each virtual environment contained two to five flying objects to be detected. The sky, lighting, and weather were varied to resemble real world photographic conditions. The result is a set of virtual images which closely emulate data obtained in real-world environments under varying conditions. Example sea, farm, and neighborhood environments may be found in Figures 2.1, 2.2 and 2.3.

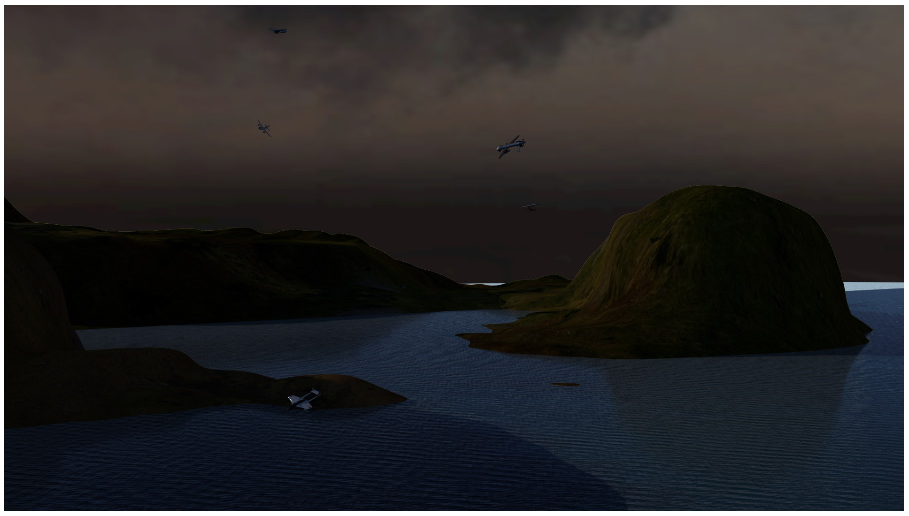


Fig. 2.1.: Virtual Sea Environment

Generating these virtual environments required extensive labor and attention to detail. Initially, the MAS team rendered each virtual environment sequence in the form of a short film 1-5 seconds in duration (approximately 30 to 150 frames or object instances per sequence). Although the network architecture does not evaluate the data in a time-series context, the time-series feature was still present in the data and there was an inherent continuity between object instances.

The virtual environment images may be rendered at arbitrarily high quality and have complex features; an evaluation of what the network was capable of analyzing, the abilities of the render farm, and the number of GPUs were all factors in the decision to render each image once at 4k, then downsample to 1k images. However,

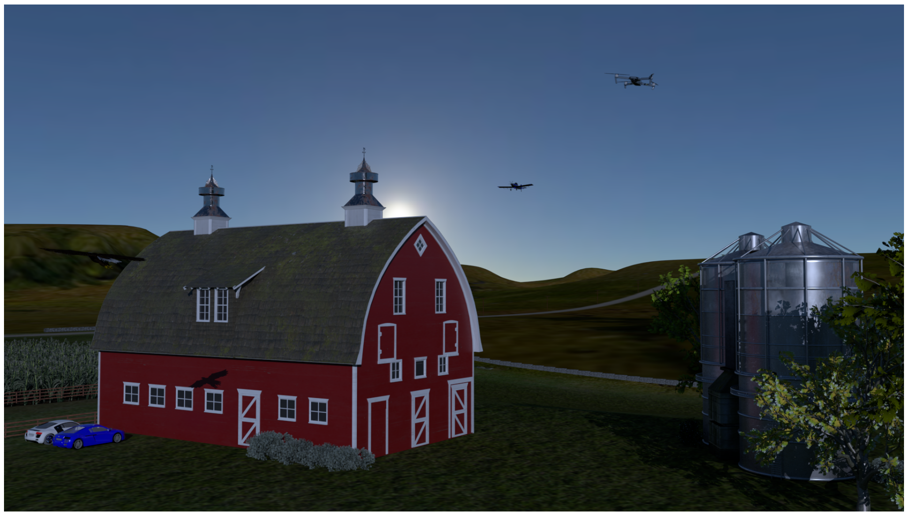


Fig. 2.2.: Virtual Farm Environment

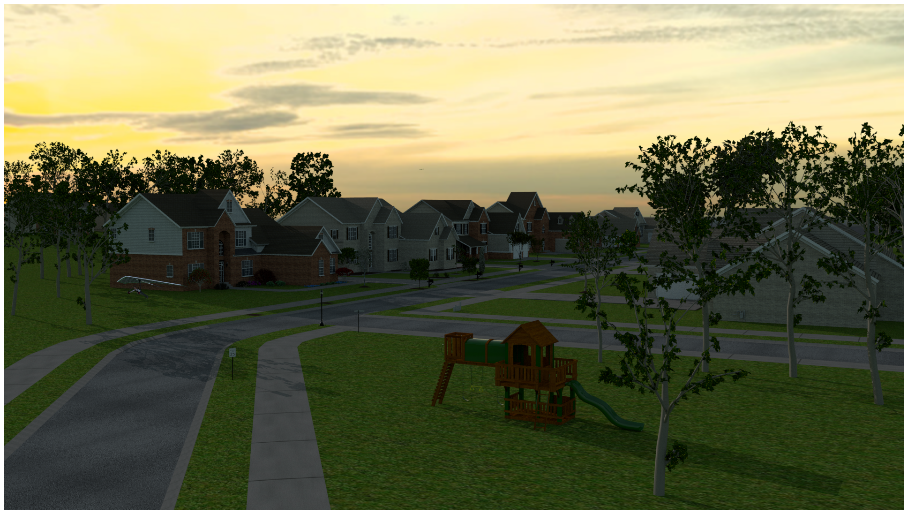


Fig. 2.3.: Virtual Neighborhood Environment

even with concessions as to image quality and scene complexity, it became difficult to generate a large amount of training data in an efficient amount of time. Accordingly, a second approach was required to generate bulk synthetic data.

In the second approach, a single object corresponding to one of the network classes (e.g. airplane or kite) was modeled in Maya, then imported into an empty environment devoid of any real-world context. The 12-camera rig shown in Figure 2.4 was then placed around the object, then rotated 360-degrees about the x-y plane while the object was rotated 360 degrees around the z-axis in an attempt to capture every possible orientation of the object. These 360-degree captures were then rendered with transparency behind the object. The result is a set of data containing a single object at every possible angle and orientation, but without any context.

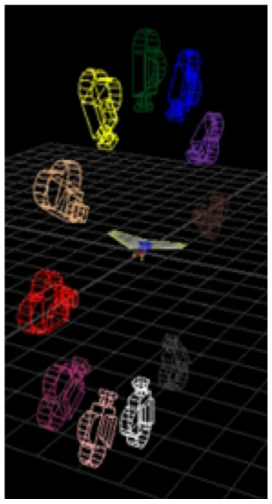


Fig. 2.4.: 360 Camera Rig used to generate bulk synthetic data of an object in every orientation. The hang glider is shown in the center. The black background is not rendered with the object.

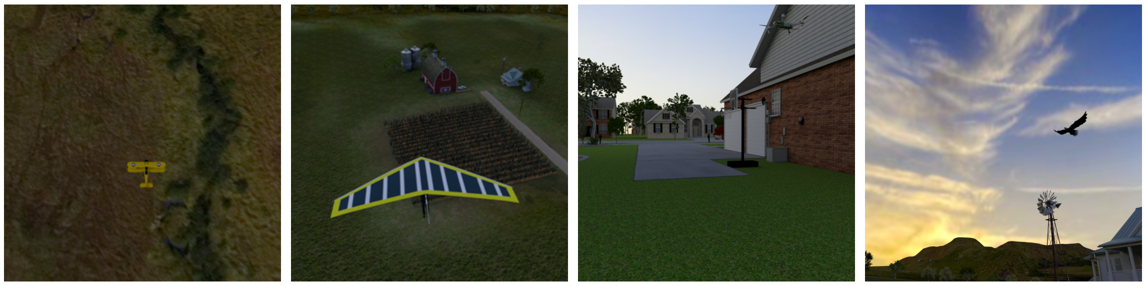
To obtain context for each object, high quality HDR images of various real-world and virtual scenes were imported into Maya separately, captured with the 360-degree camera rig, and rendered to correspond exactly with the orientation of the 360-degree renders of the object. These 360-background images could then be combined with various 360-model images to create an extensive set of mix-and-match training images. Any object could be paired with any background, and lighting adjustments could be easily handled in post-processing. As an example, Figure 2.6 shows the same grass-sky virtual background rendered with several different objects in various orientations captured by the 360-rig.

The 360-rotation approach significantly decreased the amount of craftsmanship required to generate the virtual environments of synthetic data. The comprehensive nature of the 360-rotation data encompasses the novelty and diversity of the hand-crafted sequence through sheer volume. What the 360-rotation data lacks in environmental complexity is recovered by including the virtual environment images in the final training set.

The result of combining the complex virtual environment images with the bulk 360-rendered images is a large data set of 10,116 images, denoted as RGB*, consisting of 8,640 rotational images (denoted 360-RGB*) and 1,476 virtual environment images (denoted VW-RGB*). To further ensure that any noise or real-world complexities absent from the virtual environment are maintained during transfer learning, an additional 500 real-world images were sourced from various photo repositories on the internet with a Creative Commons license. This set of extended real-world images is denoted as the RGB data set. The total data set used for training is then RGB+RGB*, and consists of 10,616 training images.

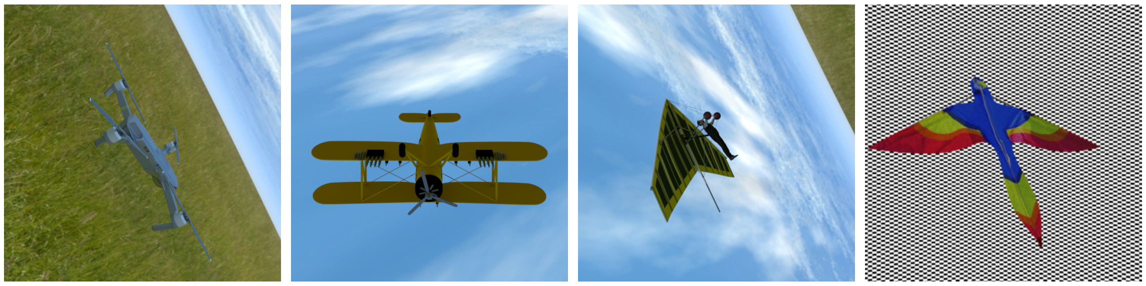
2.2.1 Models and Classes

Five objects were chosen as classes for testing and training with the transfer learning approach: plane, kite, glider, quadcopter and eagle. These objects were chosen for the following criteria: First, relevance to the real-world application of discriminating between similar flying objects. Second, similarity and dissimilarity between objects. For example, the kite was purposely modeled to resemble a bird to force a more rigorous classification between kites and birds; kites and hang-gliders share a two-dimensional planar structure which makes the front profile views very similar, etc. An example of four of the five objects with various virtual environments is shown in Figure 2.5. Of these five classes, plane, kite, and bird ("eagle") preexisted in MS COCO as labels 5, 34, and 15 respectively.



(a) (b) (c) (d)

Fig. 2.5.: (a) Plane in Farm Virtual Environment, (b) Glider in Farm Virtual Environment, (c) Quadcopter in Neighborhood Virtual Environment, (d) Eagle in Farm Virtual Environment



(a) (b) (c) (d)

Fig. 2.6.: (a) Quadcopter in 360-Rotation Virtual Environment, (b) Plane in 360-Rotation Virtual Environment, (c) Hang Glider in 360-Rotation Virtual Environment, (d) Kite in 360-Rotation Finely Checkered Environment

2.2.2 360-Degree Rotations and Various Backgrounds

Two backgrounds were implemented in the 360-RGB* data. The first background consisting of grass and sky has already been shown in Figure 2.6(a)-(c), and was used in the plane, glider, quadcopter, and eagle classes.

For the kite class, a finely checkered background was chosen, with the hypothesis that as the image is processed by the various convolutional layers, the checker would eventually be "blurred" away and encourage the network to focus on the features of the kite, rather than on any background features. An example of the kite on the checkered background is shown in Figure 2.6 (d).

2.3 Data

The RGB and the VW-RGB* data was hand labeled with boxes (top, bottom, width, height). The 360-RGB* data was labeled using an automated blob detect on the depth map to generate polygons (this algorithm is discussed further in Chapter 3.2.1). The automated approach to labeling data with polygons was intended to improve the quality of the masks generated by the output layers.

The RGB+RGB* data set is shown broken down by number of instances per class in Table 2.1. Note that the total number of instances is higher than the total number of images due to multiple instances per image. The test set consisted exclusively of the real-world RGB images discussed in Section 2.5.1.

Table 2.1.: RGB+RGB* Training and Testing Data Instances per Class

RGB+RGB* Data		
	Training Instances	Test Instances (from RGB Test Data Set)
Plane	2083	51
Glider	2527	101
Kite	2161	43
Quadcopter	2225	65
Eagle	1953	96

2.4 Methodology

The Mask R-CNN network with the ResNet-101 backbone was initialized using the MS COCO weights excluding only the head, FPN, and mask sub-networks as shown in Figure 1.2. The sub-networks were randomly initialized and allowed to train for 25 epochs with the parameters shown in Table 2.2 as discussed in Section 2.5. An epoch was defined as the completion of a number of steps equal to the total data set divided by the number of images per GPU minibatch. Training was repeated ten times each on the RGB data set and RGB+RGB* data set.

Table 2.2.: RGB+RGB* Network Training Parameters

RGB+RGB* Training Details	
GPU	GE Force GTX 1080
IMAGES_PER_GPU	4
IMAGE_MIN_DIM	1024
IMAGE_MAX_DIM	1024
STEPS_PER_EPOCH	2738
DETECTION_MIN_CONFIDENCE	0.80
VALIDATION_STEPS	2738

Early training attempts included randomly partitioning the VW-RGB* data (excluding the 360-RGB* data) into 50:50 training-testing sets, so that the test set included 50% of all available virtual environment images. However, the virtual environment test data consistently performed poorly during testing. For example, in a given 6 frame image sequence from the 360-RGB* set with only small difference in object orientation, frames 1, 3, and 5 would be in the training set and frames 2, 4, and 6 would be in the testing set. The network would have perfect recall on frames 1, 3, and 5, but fail to find a single detect in frames 2 and 4. Within the same testing set, the network would perform significantly better on all RGB images than on all RGB* images. The conclusion is that the VW-RGB* data may be more challenging for the network than the RGB data.

As a result, all available VW-RGB* data was included during training and the network was evaluated solely by its performance on the RGB test data in Table 2.2. Not only does this more-closely align with the network's actual intended application, the inclusion of complete RGB* time-series significantly improved the network's overall performance on RGB data. The results for training on all available virtual environment data and testing exclusively on real-world data are presented in Figure 2.8, where the solid colors represent the RGB test results from RGB+RGB* training.

The architecture was trained again using exactly the same approach described above, but with only the smaller 500 image RGB subset from the original RGB+RGB* training set. These training parameters are presented in Table 2.3, and the results presented in Figure 2.8 for comparison.

Table 2.3.: RGB Network Training Parameters

RGB Training Details (No RGB* data)	
GPU	GE Force GTX 1080
IMAGES_PER_GPU	4
IMAGE_MIN_DIM	1024
IMAGE_MAX_DIM	1024
STEPS_PER_EPOCH	125
DETECTION_MIN_CONFIDENCE	0.80
VALIDATION_STEPS	125

2.5 Transfer Learning with Synthetic Data

The Microsoft Common Objects in Context (MS COCO) data set was released in 2015 as a response to the growing need for extensive, complex image training data for real-world applications [56]. In its totality, the data set consists of 2.5 million per-instance labels across 328k RGB images of 91 distinct objects [56]. MS COCO emphasized multiple objects and multiple classes per image, with an average of 10k instances per object class [56]. The ResNet pre-trained weights chosen for this work used 80 of the original 91 object classes.

In the following sections, the RGB and RGB* test sets are evaluated using MS COCO weights in the Mask R-CNN architecture without transfer learning.

2.5.1 Baseline Results with RGB Real World Data

To establish a baseline comparison for the various training approaches presented in this work, the RGB (real-world only) test set was evaluated using the original MS COCO trained network with a ResNet-101 backbone (no transfer learning). The RGB test set includes three classes pre-existing in the MS COCO weights (eagle/bird, airplane/plane, and kite), and two new classes (glider and quadcopter). The testing results with a confidence threshold >80% are presented in Table 2.4.

Table 2.4.: Test results from MS COCO weights and the RGB test data set without network training (transfer learning) on the RGB(*) training data set.

Ground Truth/Network Result	**eagle**	**plane**	**glider**	**quadcopter**	**kite**
bird	102	0	23	1	1
airplane	0	57	30	43	1
person	2	15	72	67	3
kite	0	0	67	0	62
car	0	8	0	11	0

continued on next page

Table 2.4.: *continued*

Ground Truth/Network Result	**eagle**	**plane**	**glider**	**quadcopter**	**kite**
umbrella	0	1	9	0	0
tennis racket	0	1	0	2	0
surfboard	0	1	2	0	0
skis	0	1	1	0	0
backpack	0	0	1	0	1
banana	1	0	0	0	0
boat	1	0	0	0	0
knife	2	0	0	0	0
chair	0	2	0	0	0
sink	0	2	0	0	0
stop sign	0	1	0	0	0
motorcycle	0	0	2	0	0
Frisbee	0	0	1	0	0
dog	0	0	1	0	0
truck	0	0	1	0	0
bicycle	0	0	0	8	0
traffic light	0	0	0	6	0
baseball bat	0	0	0	1	0
fire hydrant	0	0	0	1	0
cell phone	0	0	0	1	0
skateboard	0	0	0	1	0
dining table	0	0	0	1	0
refrigerator	0	0	0	1	0
bottle	0	0	0	1	0
NO DETECTION	0	1	4	6	0

The baseline results show that the network behaves predictably well on pre-existing classes, with few missed detections or misclassifications. The two new classes—-glider and quadcopter—-are strongly represented by the other three classes in the test set, implying that they share a large number of features with the pre-existing classes (eagle, plane, and kite). It is also worth commenting on the diversity of the misclassifications. Given the reasonably high confidence threshold, these misclassifications imply that the real-world image data contains strong features that are present in the network, but not associated with the RGB class definitions.

Transfer learning with MS COCO and the RGB test set may then be thought of as the attempt to divide the features in the two pre-existing classes among five test classes, and re-assign features from an additional 26 MS COCO classes to five test classes.

2.5.2 Baseline Results with RGB* Synthetic Data

It is useful to establish a baseline comparison for the various training approaches in Chapters 2 and 3 using the RGB* (virtual-world only) test set. The test set was evaluated in a manner identical to the one above. The RGB* test set omits the kite class, resulting in two classes pre-existing in the MS COCO weights (eagle/bird and airplane/plane), and two new classes (glider and quadcopter). The testing results with a confidence threshold >80% are presented in Table 2.5.

Again, the MS COCO weights perform well on the pre-existing classes, and the two new classes are strongly represented by the two pre-existing classes in the test set. In contrast to Table 2.4, there are relatively few classes represented in the misclassifications and there are a higher number of missed detections.

Table 2.5.: Test results from MS COCO weights and the RGB* test data set without network training (transfer learning) on the RGB* training data set.

Ground Truth/Network Result	**eagle**	**plane**	**glider**	**quadcopter**
bird	214	0	51	5
airplane	10	246	176	304
kite	10	0	273	0
person	14	18	135	2
umbrella	0	0	14	3
surfboard	0	0	13	0
traffic light	0	1	0	3
bear	1	0	0	0
NO DETECTION	10	0	39	0

2.5.3 Results of Transfer Learning RGB+RGB* Training

The mean F1-score per epoch along with the standard variation in F1-score for network test results on RGB data is presented in Figures 2.7. In the plots for four of the five classes (excluding eagle), the network converges more quickly with the larger RGB+RGB* data set than with RGB data alone and achieves an F1-score equal to or greater than the F1-score for RGB data alone. The only class to not show improvement, eagle, pre-existed in the MSCOCO data set as "bird" and already had an F1-score >0.95.

Although the network performed well on the RGB test set, the maximum of the averaged F1-scores per epoch for the RGB-trained network is surpassed by the maximum averaged F1-scores per epoch for the RGB+RGB*-trained network on four out of the five classes as shown in Figure 2.8. The average maximum F1-score of all classes and all epochs for the networks trained with synthetic data is $F_1^* = 0.91$, compared to $F_1 = 0.89$ for the networks trained exclusively with real data, and the standard deviation of the maximum mean F1-score for synthetically trained networks is $\sigma_{F1}^* = 0.015$, compared to $\sigma_{F1} = 0.020$ for the networks trained exclusively with real data.

Example mask results from training are shown in Figure 2.9. It may be possible to further improve mask results by freezing all layers except those generating the mask. This would allow the mask to continue refinement without over-training the classification output layers.

2.5.4 Impact of Various Synthetic Backgrounds

The effect of the convolutional layers in the ResNet backbone on image backgrounds are examined using weights from the RGB+RGB* trained network. Two images, shown in Figure 2.10 were used to test the ResNet-101 backbone. The kite image series was rendered with a checkered background to test if 1) the background could be convolved away entirely and 2) if an abstract background negatively im-

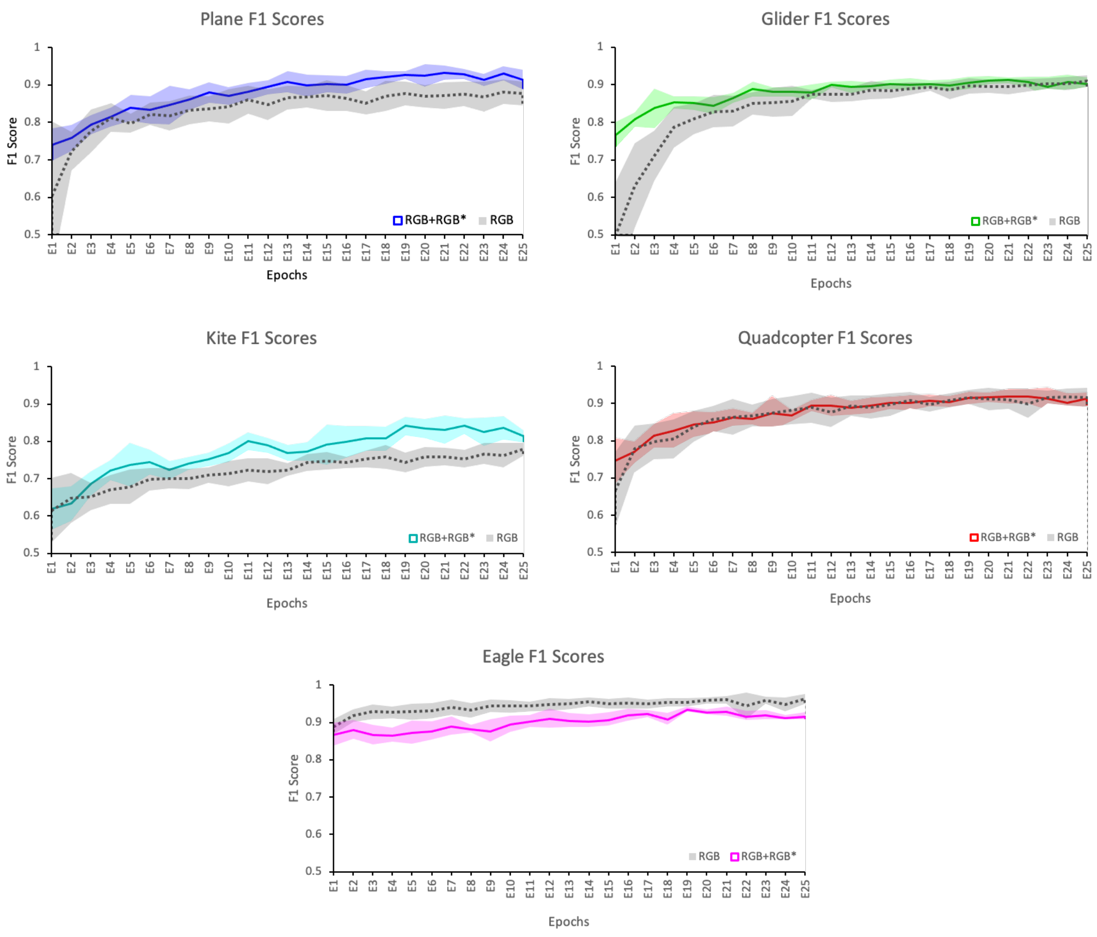


Fig. 2.7.: Comparison of F1-scores from RGB+RGB* trained network and RGB trained network from Section 2.5.1 for various classes using RW data. The lines represent the mean F1 value from ten repeated trainings at each epoch, while the shading around the line shows the standard deviation of the mean F1 value. The solid line is the RGB+RGB* training results; the dotted, gray line is the RGB training results.

pacted the network's ability to detect an object—that is, if background contextual clues were helpful during training. The checkered background is a high-frequency but single frequency signal compared to the sky-grass background which contains a distribution of both low and high frequencies as shown in the FFT information in Figure 2.10.

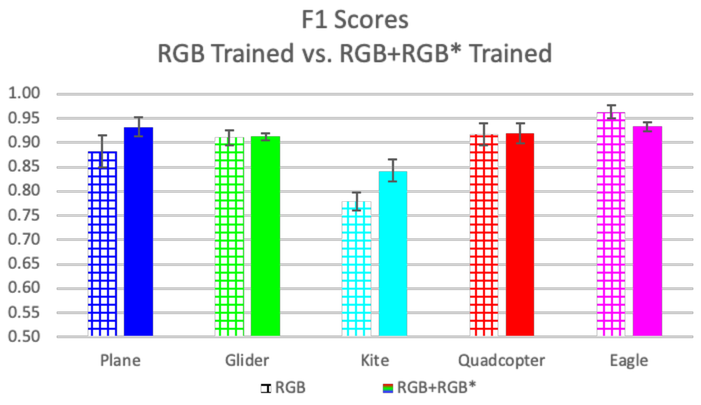


Fig. 2.8.: Comparison of F1-scores from RGB+RGB* trained network and RGB trained network from Section 2.5.1. The maximum mean F1-score from ten repeated network trainings is presented with error bars showing the max and min standard deviation in the max score. The RGB+RGB* trained network beats the RGB trained network in four out of five classes, including the three classes pre-existing in MS COCO.

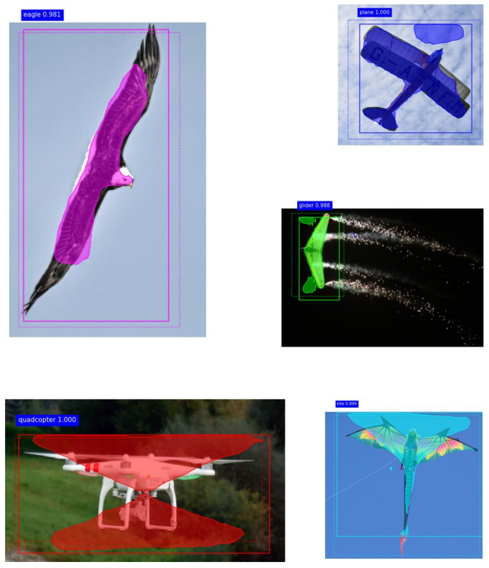


Fig. 2.9.: Example RGB test results from training with the RGB+RGB* data set. From top left clockwise: eagle [57], plane [58], glider [59], kite [60] and quadcopter [61].

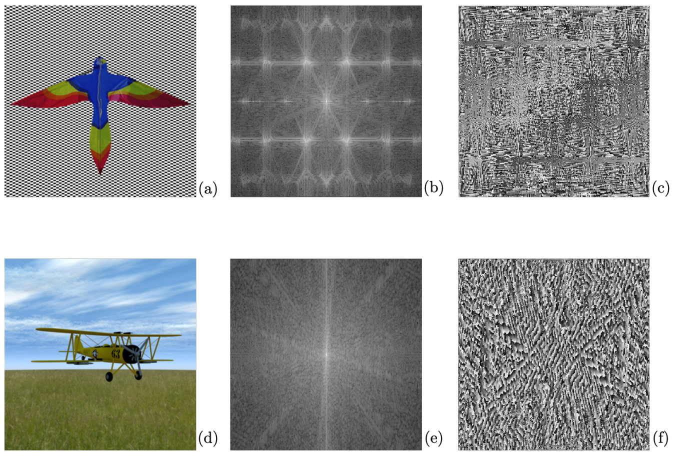


Fig. 2.10.: Two synthetic images used to test the ResNet-101 backbone with RGB+RGB* transfer learning weights. (a) Input synthetic kite image with checkered background. (b) Magnitude information from the FFT of image (a). (c) Phase information from the FFT of image (a). (d) Input synthetic plane image with sky-grass image. (e) Magnitude information from the FFT of image (d). (f) Phase information from the FFT of image (d).

It is important to note that the entire image enters the network through the layer *conv1*, rather than as individual RoIs taken from the image. Furthermore, since both input images are already square with input dimensions 320x320x3, the *zero_padding* layer immediately preceding *conv1* has no effect on the aspect ratio of the image—leaving the image background undisturbed—but does up-sample the input to 1030x1030x3. Finally, although the first sixteen layers of the output tensor are returned in each of the following figures, it should not be assumed that the layer samples are consistent from image to image (that is, the sample in the top left corner of Figure 2.12 should not be assumed to be a direct alteration of the sample in the top left corner of Figure 2.11).

The output of *conv1* is a 512x512x64 tensor; Figure 2.11 shows the first sixteen 512x512 layers of the tensor returned for the kite and plane images. In all of these images, the background is clearly discernible from object and recognizable from the original input images.

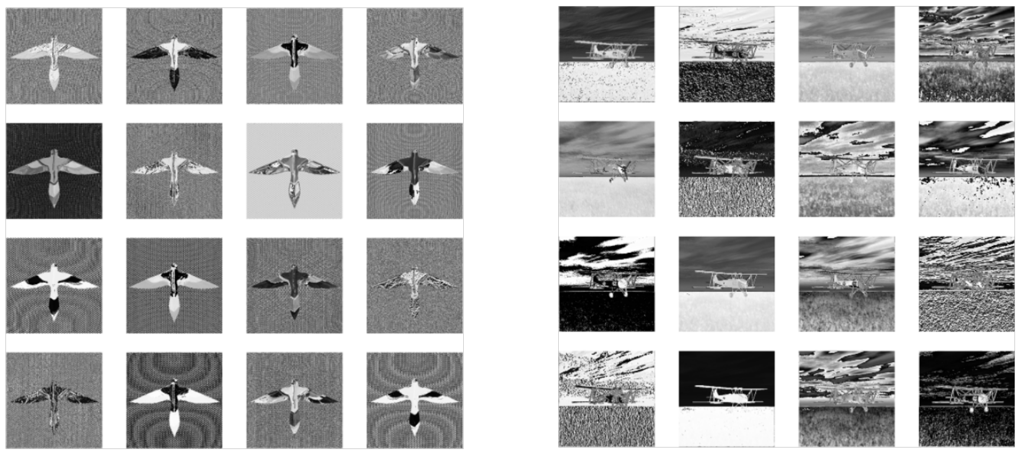


Fig. 2.11.: RGB* output of layer *conv1* in the ResNet-101 backbone showing the first sixteen 512x512 layers from the 512x512x64 tensor. Left: Kite. Right: Plane.

The output tensor was examined again at the output of layer *res2c_out*, where the first major residual block consisting of ten *Conv2D* layers ends. The output tensor now has dimensions 256x256x256, and again the first sixteen 256x256 layers are shown in Figure 2.12. The background still survives in both the checkered and sky-grass form, and it also becomes apparent from looking at the red-blue-yellow stripes on the kite that certain convolutional kernels are responsible for detecting features based on color.

At the output of layer *res3d_out* which ends the second large residual block consisting of an additional twelve *Conv2D* layers, the tensor dimension is 128x128x512 and aliasing effects become visible as shown in Figure 2.13. However, the background features still survive in both the checkered and sky-grass environment.

The largest residual block unit consisting of 70 *Conv2D* layers ends with layer *res4w_out*, and outputs the tensor with size 64x64x1024 partially shown in Figure 2.14. Although the sampling of sixteen layers from the 1024 in the tensor is small, the background texture seems to be largely gone, and the primary feature remaining seems to be edges. This is to be expected, however even a superficial glance hints that the edges of the kite are more uniformly presented in the sampling presented in

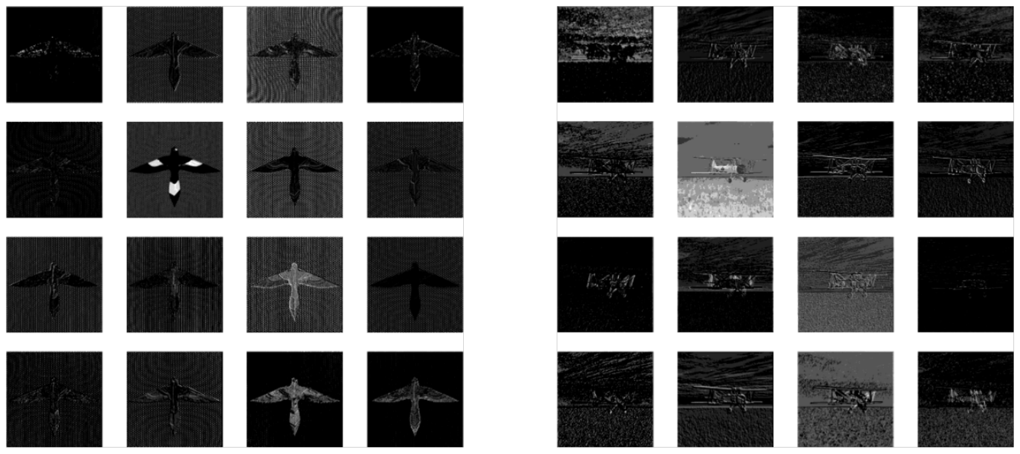


Fig. 2.12.: RGB* output of layer *res2c_out* in the ResNet-101 backbone showing the first sixteen 256x256 layers from the 256x256x256 tensor. Left: Kite. Right: Plane.

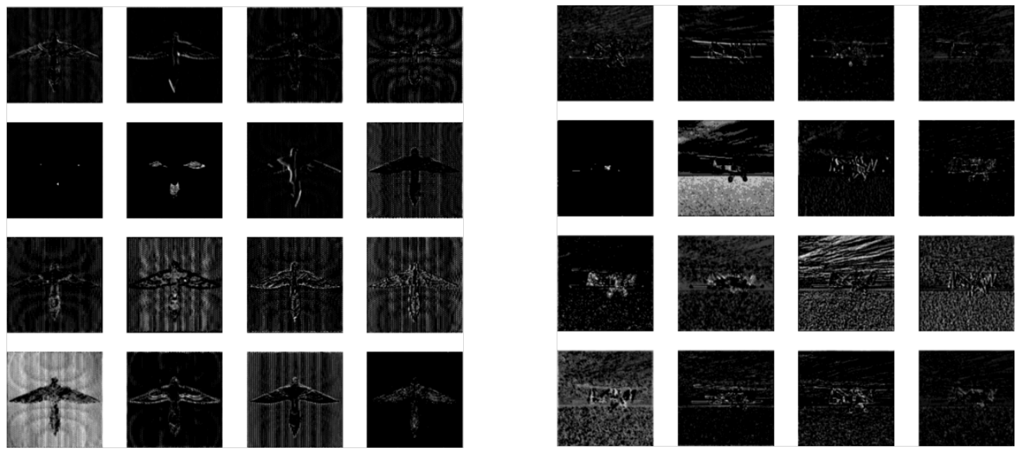


Fig. 2.13.: RGB* output of layer *res3d_out* in the ResNet-101 backbone showing the first sixteen 128x128 layers from the 128x128x512 tensor. Left: Kite. Right: Plane.

Figure 2.14 than the edges of the plane. The sampling from the plane also contains more noise in the background, indicating that the network is training on background features.

The output of the final ten convolutional layers comes from layer *res5c_out*, and is shown in Figure 2.15. At this point, the tensor has dimensions of 32x32x2048 so only the largest features should remain. Of the sixteen tensor arrays sampled, the

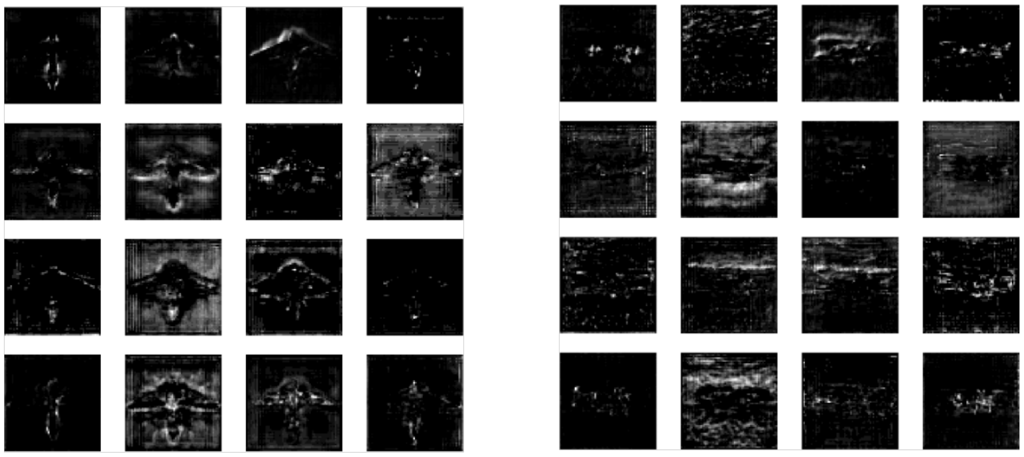


Fig. 2.14.: RGB* output of layer *res4w_out* in the ResNet-101 backbone showing the first sixteen 64x64 layers from the 64x64x1024 tensor. Left: Kite. Right: Plane.

kite only remains recognizable in row 3, column 2. The plane is not recognizable in any of the samples in Figure 2.15.

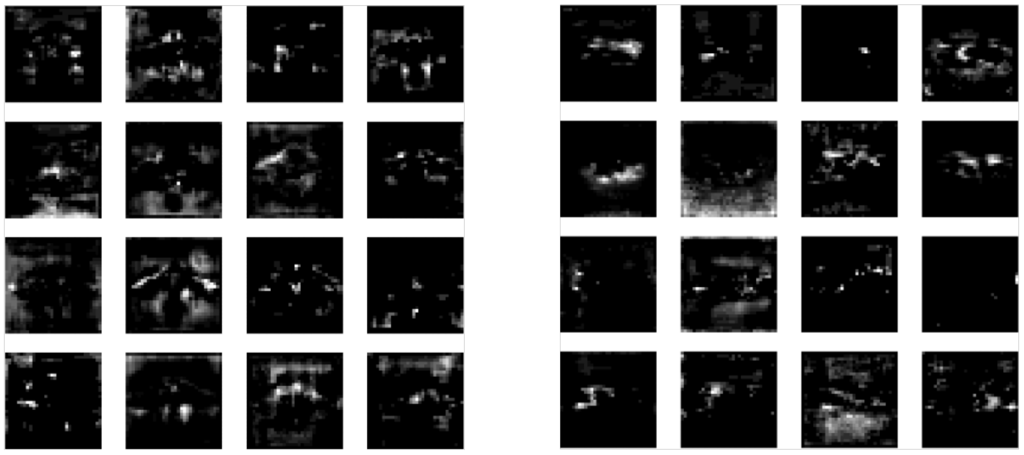


Fig. 2.15.: RGB* output of layer *res5c_out* in the ResNet-101 backbone showing the first sixteen 32x32 layers from the 32x32x2048 tensor. Left: Kite. Right: Plane.

Qualitatively, the checkered background seems to have little effect on the convolutional feature maps compared to the sky-grass background. When the input image tensors reach the *res5c_out* layer, both backgrounds are equally unrecognizable, and since only sixteen of the 2,048 tensor layers are sampled, no conclusions should be drawn as to whether the high-frequency background has more or fewer features than the lower-frequency sky-grass background. Quantitatively, over the ten repeated training runs with RGB+RGB* data, the max F1-score for the “kite” class improved by 0.062 and while the standard deviation for the “kite” class decreased by 9.5E-05 (effectively zero change) compared to the RGB training. For the “plane” class, the max F1-score improved by 0.050 with RGB+RGB* training while the standard deviation for the “plane” class decreased by 6.4E-03 (effectively zero change).

The similarity in training results seems to suggest that for this data set the difference in the checkered background had no effect compared to the sky-grass background. However, when the synthetic data set was created by the MAS team, the virtual environments required significantly more labor than the checkered background. Therefore, it may be possible to reduce the effort required to produce an effective synthetic data set by using context free backgrounds during training.

2.6 Conclusion

The original MS COCO data set used to create the initialized weights consisted of over 328k real-world images [56], and was the source of all the features used during classification. Accordingly, the role of the synthetic data was limited to assisting in the reassignment of the pre-existing features to the new classes during transfer learning. When compared with the baseline results presented in sections 2.5.1 and 2.5.2, it is clear that the synthetic data was successful in reassigning features to new classes.

With a dedicated GPU and the AutoDesk Maya software, the final rendering of 10k 320x320 360-degree rotational RGB* images may be accomplished on a desktop in less than 12 hours. The final rendering of virtual-environment 320x320 images may be accomplished in a similar time scale of hours. This would seem to suggest that suitable synthetic data is easily manufactured and produced. However, there is a considerable time and labor investment up front in the modeling, texture, and construction of models and virtual environments that should not be discounted. For 360 rotational data, once the modeling and texturing is completed, generating new data sets is indeed straightforward. In contrast, virtual-environment data has an artistry and complexity that makes it difficult to mass produce, but adds value to the training set.

The uniqueness of the checkered background in the 360-degree kite RGB* data did not significantly affect the improvement in RGB+RGB* over RGB training–nor did it significantly detract from the testing result; the gains are comparable to those with the sky-grass background. This may imply that, in situations where an adequate representation of a real-world environment does not exist, a simple abstract substitute may be a viable option. The use of context-free backgrounds in 360-RGB* data may also be useful in further reducing the effort in creating synthetic data sets.

The final training data set used during transfer learning consisted of a ratio of one real-world image to approximately twenty synthetic images, and one virtual environment image to approximately six rotational images. The ultimate impression is that the bulk of examples, rather than the overall quality (synthetic vs real, environmental background vs checkered, etc.), enabled the improved F1-scores. More data, it would seem, is still be better in this case *as long as the class pre-existing in MSCOCO had room for improvement.* The eagle/bird class in MSCOCO already had a very high F1-score, and the addition of synthetic data detracted from the pre-existing data set. Accordingly, future consideration should be given to the ideal balance of real-world images, synthetic rotational images, and synthetic environmental images to achieve optimum network results with as little active labor on the data set as possible.

3. TO Z OR NOT TO Z: IMPACT OF INCLUDING DEPTH DATA ON TRAINING

3.1 Introduction

Although several large data sets are available that feature real-world (RW) depth images paired with RGB images [62–64], they tend to focus on either outdoor data or household objects; these classes do not overlap with the classes established early in the project. Furthermore, since it is not the purpose of this work to evaluate network architecture performance, but instead to consider how specific data features affect training. Performing benchmark tests with previously published RGBZ data is beyond the scope of this project. Accordingly, these data sets were not considered for this work.

Instead, we focus on the role of synthetic depth data during training, with a small amount of fabricated RW depth data paired to RGB images for comparative purposes in Chapter 2. Section 3.2 discusses the creation of the depth component for the extension of RGB+RGB* to RGBZ+RGBZ* in detail. An attempt was then made to duplicate the studies presented in Chapter 2 with transfer learning on the MS COCO weight initializations; these results are presented in Section 3.4.

3.2 Data

The RGB+RGB* data set discussed in Chapter 2 was expanded with depth maps into the RGBZ+RGBZ* data set and used for the experiments presented in this chapter. For the RGBZ* data discussed in Section 3.2.1, depth maps were generated in AutoDesk Maya by the Media Arts and Sciences (MAS) team. The RGBZ data consists of hand-edited depth maps as discussed in Section 3.2.2. The total

RGBZ+RGBZ* data set is then exactly the images presented in Table 2.1, but with an added depth channel.

3.2.1 Synthetic Depth Data

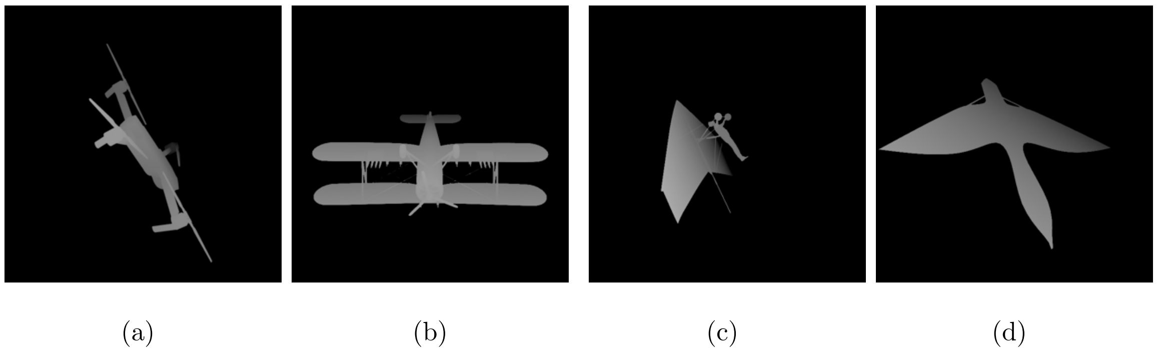
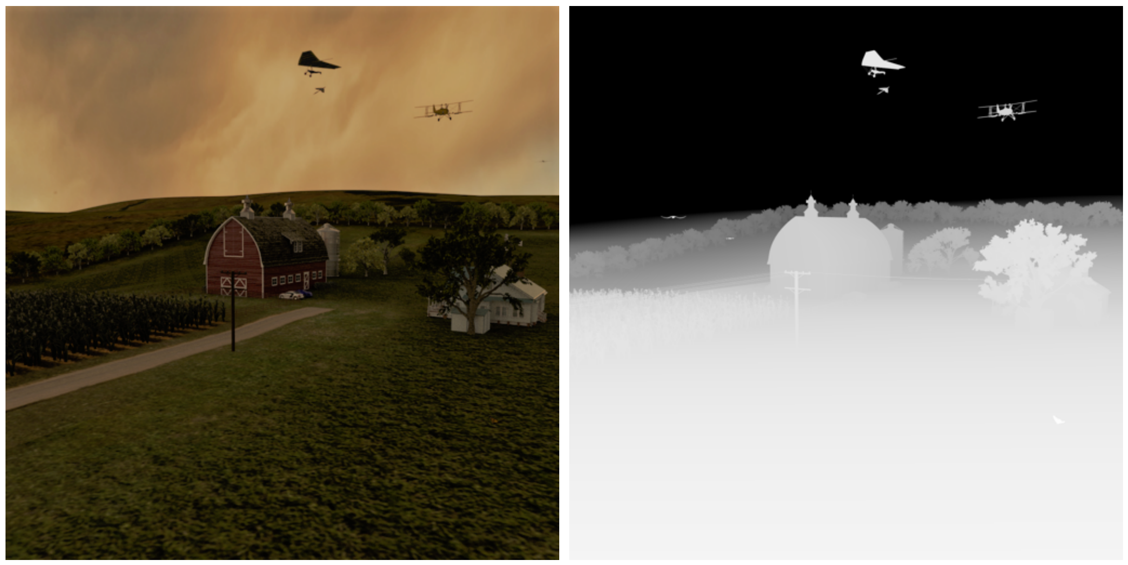


Fig. 3.1.: Depth maps corresponding to the RGB* images presented in Figure 2.6. (a) Quadcopter in 360-Rotation Virtual Environment, (b) Plane in 360-Rotation Virtual Environment, (c) Hang Glider in 360-Rotation Virtual Environment, (d) Kite in 360-Rotation Finely Checkered Environment

Maya Autodesk is capable of rendering a depth map of the distance from the object to the camera separately from rendering a color 3-channel image. The MAS team had the ability to render a synthetic depth map for every synthetic image discussed in Chapter 2. Example depth maps corresponding to the RGB* images in Figure 2.6 are shown in Figure 3.1. Autodesk Maya produces pixel values linearly scaled between zero and one, where zero is the distance at infinity and one is the camera lens. There is no constraint that the pixel values map to any RW depth unit or measurement.

The variability in depth scale is best understood by comparing the objects in the depth images in Figure 3.2 to those in Figure 3.1. For each of the depth maps in Figure 3.1, the depth value scale is optimized to preserve object features, and all other features are placed at infinity. If the depth scale in Figure 3.2 was identical to Figure 3.1, the image would appear largely black, with most objects at some great distance from the camera. Instead, the depth scale emphasizes the features of the virtual environment. Customizing the depth scale per image series enables more information to be presented to the network during training, but the variability in depth scale may prevent the network from learning an absolute depth scale.



(a) (b)

Fig. 3.2.: Example of a VW image and the corresponding image. (a) RGB* image of VW Farm environment created by MAS team. (b) Corresponding synthetic Z-depth as calculated by MAS team using AutoDesk Maya.

During depth image rendering, the zero-to-one depth scale is converted to uint8 values (0 to 255) for the output image. This places a further constraint on the depth scaling; not only does the absolute range of depth change from image to image, but the depth increments with varying step sizes as well.

Since the MR-CNN network architecture emphasizes scale invariance by implementing the FPN, the changing depth units are presumed to be handled in approximately the same manner as any other variance in object size. For any object, both variation in depth across the object (as shown in the gray scale variance for the plane in Figure 3.1) and constant depth across the object (as shown for the plane in the upper right corner of Figure 3.2) are equally features of the object, just as a plane may be colored blue or red but still be successfully classified as a plane. In practicality, the lack of consistent depth scaling admits the possibility that the data set is actually training the network to ignore the depth scale entirely, and instead emphasizing other depth map features such as edges.

Automated Labeling of 360-RGBZ* Data

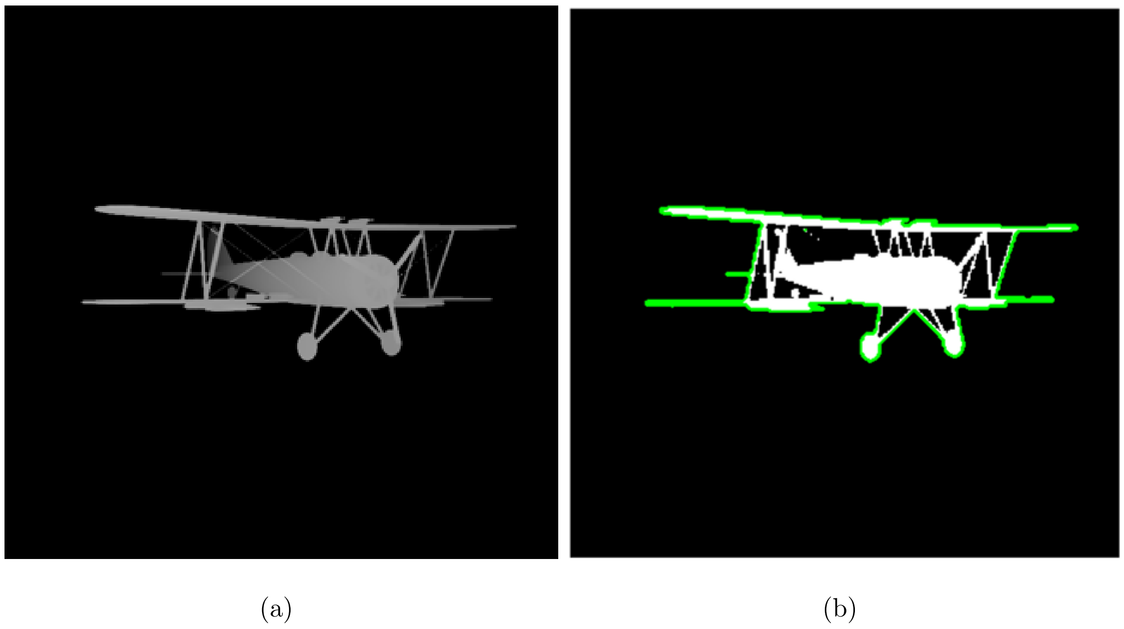


Fig. 3.3.: Effect of using the automated labeling algorithm. (a) Input depth map. (b) Output image with object boundary shown in green.

The large number of 360-RGB* images generated require labeling for use during supervised learning. Since each 360-Z* image contains a single object against a black background, the following algorithm was implemented in MATLAB's Image Processing toolbox with great success to automate the labeling process:

1. Apply an adaptive binary threshold with sensitivity=0.6 to the depth image
2. Perform boundary detect with the constraint that there are "no holes"
3. Return the pixel boundary as (x,y) pairs in a list

4. Format the pixel list as part of the ground truth label

An example input depth map and output boundary is shown in Figure 3.3. The output region is exactly the mask desired as a final output of MR-CNN. This final pixel boundary by definition is identical for the original 360-RGB* image, which means that the depth maps may also be used to label the entire 360-RGB* data set regardless of whether training actually implements depth data.

3.2.2 Real-World Depth Data

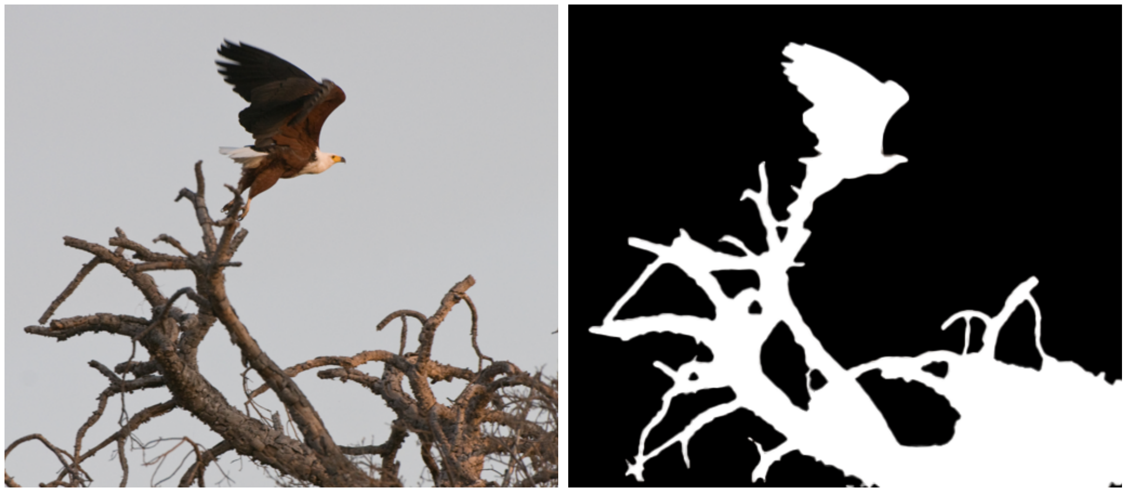
In order to approximate the work presented in Chapter 2, a RW depth image approximation was hand fabricated for each image of the RGB training set using Adobe Photoshop. Due to the input layer of the MR-CNN architecture, all input images were required to have the same number of dimensions along the third axis: a data set must therefore consist entirely of 3-channel images or entirely of 4-channel images. By implementing depth maps, the RGB images and features were able to be included in the data set of RGB(Z*) data. An example RGB image and the fabricated depth are shown in Figure 3.4.

3.3 Methodology

For the first experiment, the Mask R-CNN network with the ResNet-50 backbone was initialized using the MS COCO weights excluding only the head, FPN, mask sub-networks, and initial convolutional layer *conv1* as shown in Figure 1.2. The excluded layers were randomly initialized and allowed to train with the parameters shown in Table 2.2 to provide comparison with the work of Chapter 2.

3.4 Transfer Learning with Depth Data

Early results showed that the network struggled to learn all five classes from the RGBZ+RGBZ* data set. In particular, a trade-off in performance was observed be-



(a) (b)

Fig. 3.4.: Example of RGB image [65] from “eagle” class RW training set and hand-fabricated depth map. (a) Original Image. (b) Fabricated depth map.

tween the kite class and the glider class such that training well on one class correlated with decreased performance in the other. This is the data presented in Section 3.4.1.

A reduction to four classes still did not show the performance gains expected from the inclusion of depth data. Training was repeated with the original five classes, and the network layers initialized with MS COCO weights were updated during backpropagation. These results are then compared to a network with the ResNet-50 backbone trained using RGB+RGB* data, and presented in Section 3.4.2. Further examination of the effect of the depth data is discussed in Section 3.4.3, where the effects of the depth channel are traced through the ResNet-101 backbone. Transfer learning repeatedly on the RGBZ* training/testing data set gives excellent performance on pre-existing classes, but poor performance on new classes.

3.4.1 Results of Training Heads Only

When an RGBZ* image enters the network, the MxNx3 RGB* image is concatenated with the corresponding MxNx1 Z* image to create an MxNx4 image. To adapt the network to a four-channel RGBZ input during transfer learning, the initial 2D convolutional layer *conv1* is also randomly initialized. To use the pre-trained weights for the layer would constrain it to 3-channel images since no transfer weights previously existed for RGBZ images and the MR-CNN architecture. The Matterport implementation of the MR-CNN network included documentation for implementing a fourth image channel in the form of an alpha channel; this documentation has been adapted for various depth MR-CNN implementations, including some which use a second, separate input for the depth image rather than concatenating the depth into the fourth channel [66, 67].

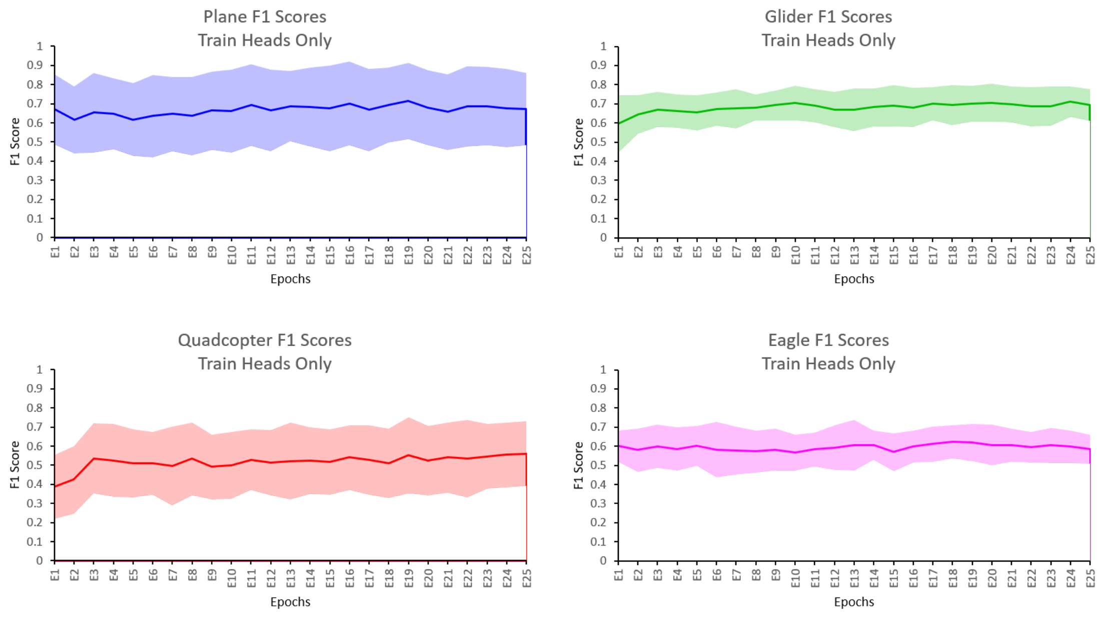


Fig. 3.5.: F1-scores by epoch for various classes when training MR-CNN heads with depth data.

Due to the back propagation method in the Matterport implementation, the *conv1* layer weights do not update during training when subsequent layers have frozen weights. The result is that, while the heads and output layers are trained, the initial convolutional layer remains frozen in its randomly-initialized state.

The MR-CNN architecture was allowed to train with these constraints for 25 epochs, and training was repeated 10 times. The results are shown in Figure 3.5. In general, the network did not perform well on the RGBZ test set, resulting in a large standard deviation in F1-score, and little improvement in F1-score as the number of epochs increases.

This data is interesting for two reasons: First, by failing to improve in any class with training on the RGBZ+RGBZ* data the way MR-CNN results improved when training with RGB+RGB* data, we see the impact of the initial convolutional layer. The network architecture sees the same input features from the first training step to the final training step. Since these input features are randomly filtered, the variation in mean F1-score from class to class can be interpreted as the native performance of the ResNet backbone at extracting features. Second, the very large standard deviation shows that there is some random initialization that performs well on the RGBZ test data without training. This confirms that the state of the network after initialization has huge impact on the final training results, even when that initialization is random.

3.4.2 Results of Relaxing Transfer Weights

Stabilizing the performance of the *conv1* layer requires allowing the network to update the weights in every layer. Since the MS COCO weight initialization provided all of the features up until this point, relaxing the network weights and tuning the architecture allowed the features intrinsic to the network to evolve to something closer to the training data. This allowed the opportunity to reintroduce the “kite” class eliminated from training during Section 3.4.1.

The network was trained for 25 epochs, training was repeated ten times, and the results are compared to RGB+RGB* trained networks (ResNet-50, heads only, no weight relaxation) in Figure 3.6.

Allowing the initial convolution layer to train clearly improves the testing results. In the majority of the five classes, the network undergoes exponential improvement

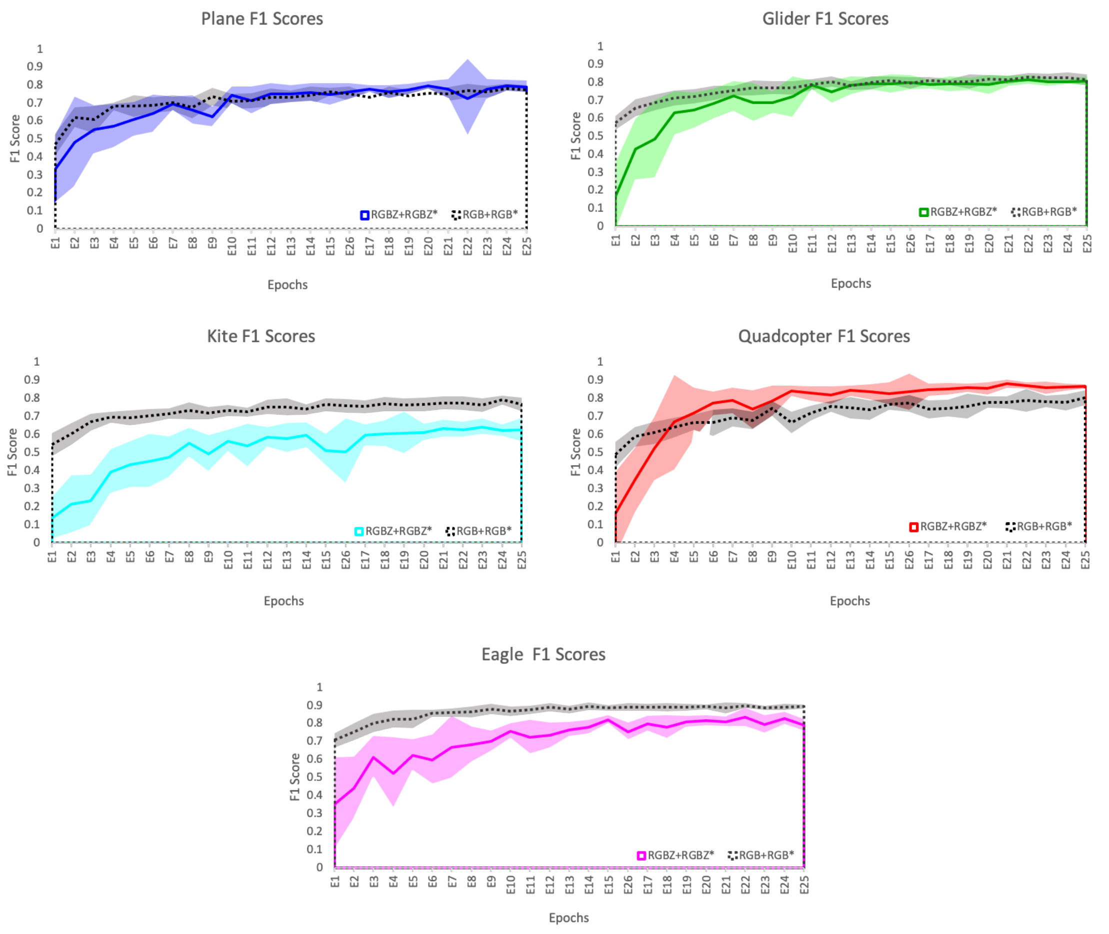


Fig. 3.6.: F1-scores for RGBZ+RGBZ* data by epoch when allowing MR-CNN MS COCO weights to relax plotted against the F1-scores by epoch for RGB+RGB* trained networks (ResNet-50, heads only, no weight relaxation.

in F1-score before converging to a value close to the RGB+RGB* values. There is also no longer a penalty for simultaneously including the “kite” and “glider” classes.

3.4.3 Impact of Depth Map on Training

Effect of Repeated Convolutional Layers on 4-channel Image

An average-performing weight set from the repeated trainings in Section 3.4.2 were connected to the MS COCO ResNet-101 backbone despite being trained with the ResNet-50 backbone. This allowed evaluation of the extreme effects of the depth image based on the assumption (due to previous RGB+RGB* results) that all necessary features for evaluating the depth maps are present in the ResNet-101 backbone despite not tuning the network weights with the depth images. The same two images from Section 2.5.4 were evaluated at the same points in the network, with results presented in Figures 3.7 to 3.11.

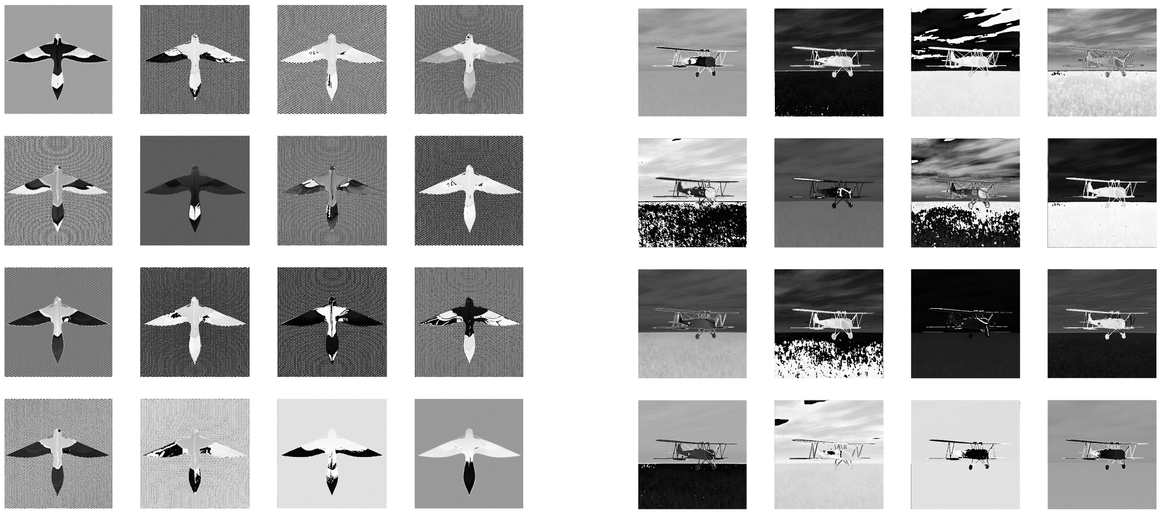


Fig. 3.7.: RGBZ* output of layer *conv1* in the ResNet-101 backbone showing the first sixteen 512x512 layers from the 512x512x64 tensor. Left: Kite. Right: Plane.

Based on the RGB* results in Figures 2.11 to 2.13, the results shown in Figures 3.7 and 3.8 are expected, with background features and textures surviving multiple convolutional layers into the network. However, the results of shown in Figure 3.9 begin to diverge from the RGB* data shown in Figures 2.13 and 2.14: background features seem to completely disappear, and the most prominent features in the tensor samples shown are edges. By the final convolutional layer in ResNet-101 shown in Figure 3.11, there are few to no features left in the tensor sample shown. It is unlikely that no object features survive in all of the other (not shown) 2,032 matrices from the 32x32x2048 dimensional tensor at layer *res5c_out*, but the effect of the depth map in the sixteen sampled is remarkable. Since the data presented in Section 3.4

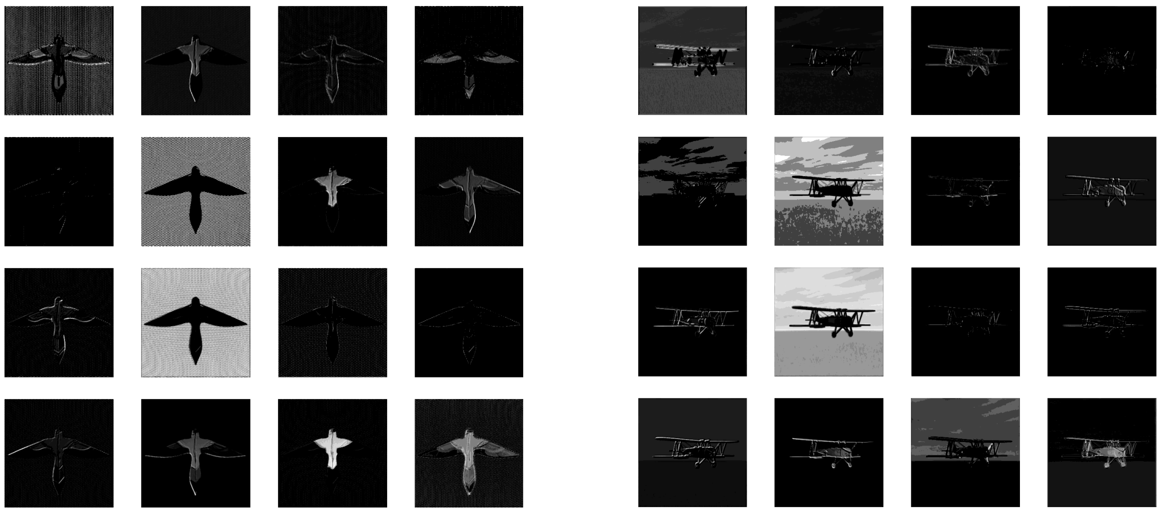


Fig. 3.8.: RGBZ* output of layer *res2c_out* in the ResNet-101 backbone showing the first sixteen 256x256 layers from the 256x256x256 tensor.
Left: Kite. Right: Plane.

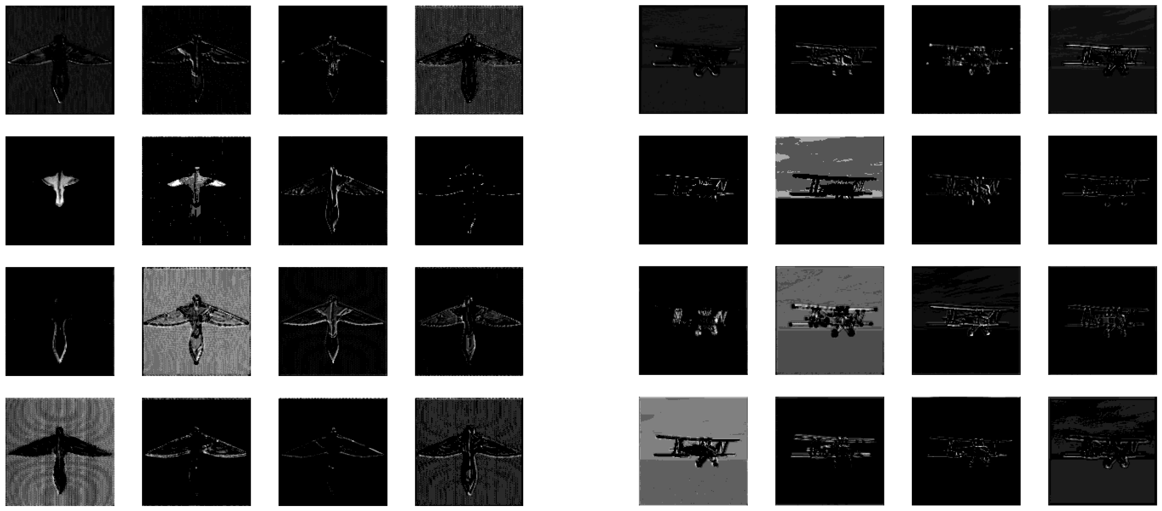


Fig. 3.9.: RGBZ* output of layer *res3d_out* in the ResNet-101 backbone showing the first sixteen 128x128 layers from the 128x128x512 tensor.
Left: Kite. Right: Plane.

implemented the ResNet-50 backbone, detections were made based on features similar to those shown in Figure 3.10.

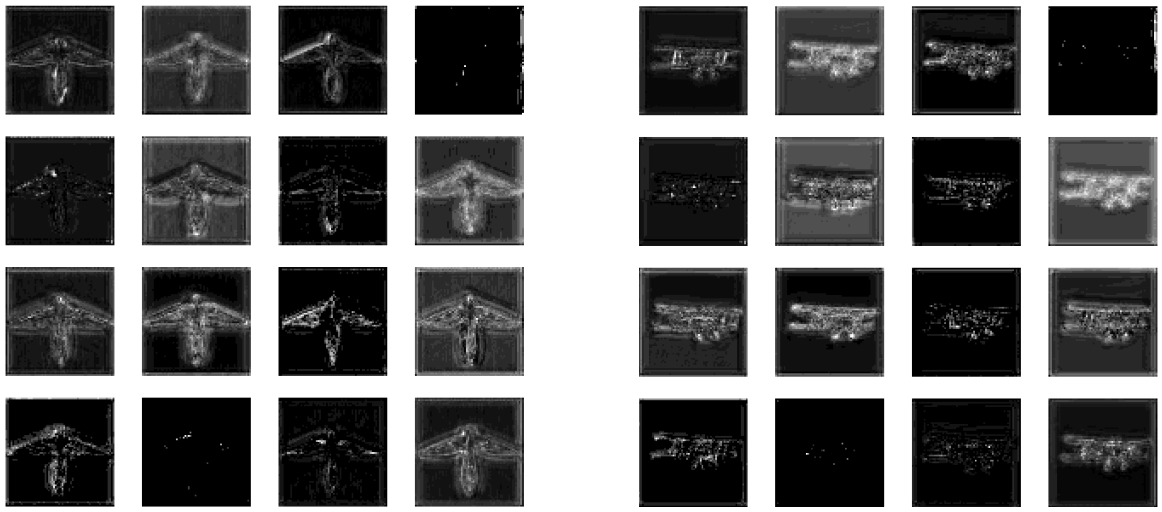


Fig. 3.10.: RGBZ* output of layer *res4w_out* in the ResNet-101 backbone showing the first sixteen 64x64 layers from the 64x64x1024 tensor.
Left: Kite. Right: Plane.

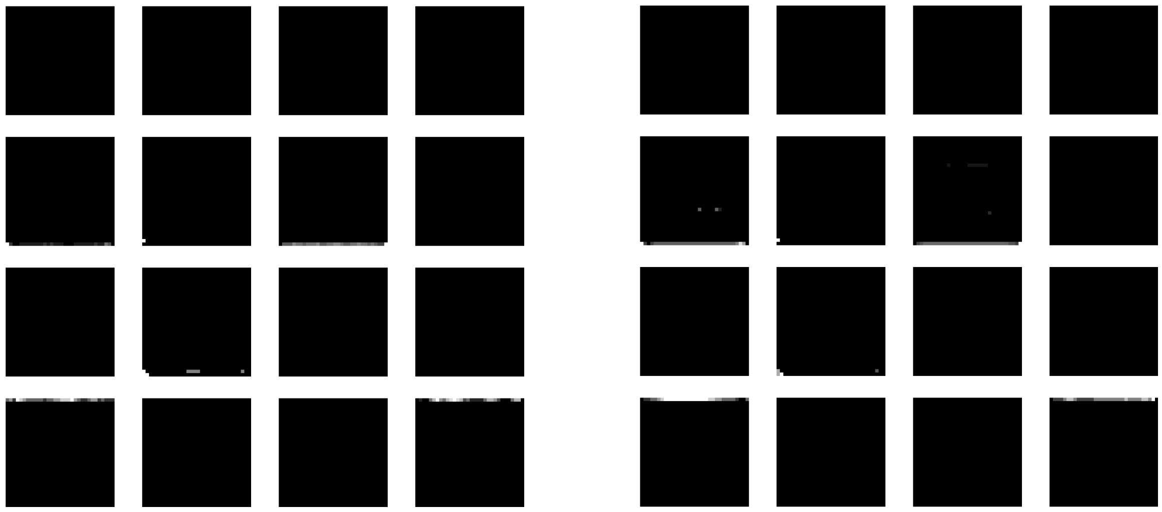


Fig. 3.11.: RGBZ* output of layer *res5c_out* in the ResNet-101 backbone showing the first sixteen 32x32 layers from the 32x32x2048 tensor.
Left: Kite. Right: Plane.

Interdependence of RGB and Z Features During Detection

To test if the network was implementing the depth contribution correctly, two example RGBZ* images were chosen, and the depth layers were swapped as shown in Figure 3.12. For illustrative purposes, the depth channel is rendered as an alpha-channel such that pixels with a 0 value in the depth image (infinite distance) are omitted from the color image. These mixed image pairs were then tested with the tuned-weight version of the ResNet-50 MR-CNN implementation. When tested with the ResNet-101 implementation, no detections resulted.

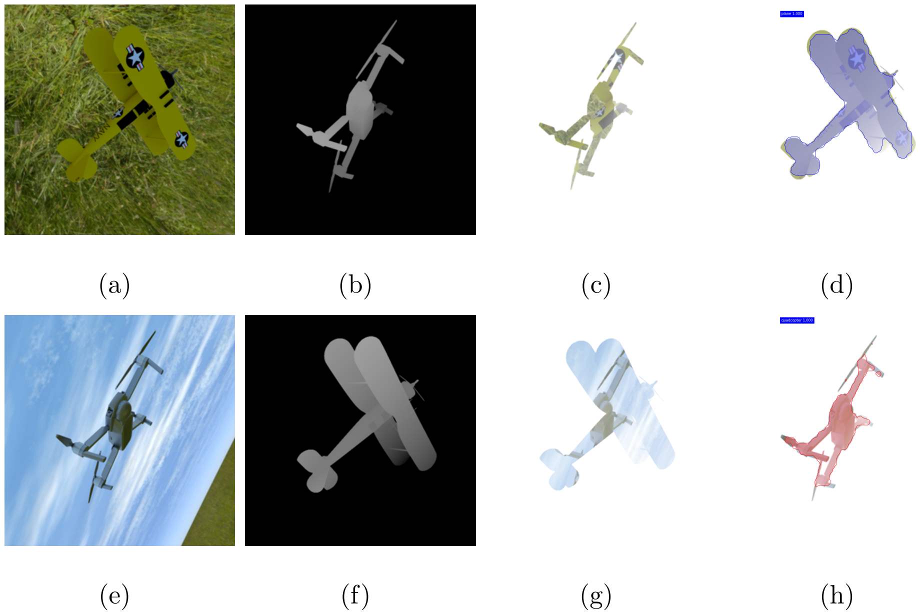


Fig. 3.12.: Creation of image pairs used to test effect of depth data during detection, where sub figures (c) and (e) were presented to the network. (a) RGB* plane. (b) Z* Quadcopter (c) RGB*(plane)+Z*(quadcopter) where the depth map effect has been interpreted as an alpha channel. (d, h) Network result on correct RGBZ* pairing. (e) RGB* quadcopter. (f) Z* plane. (g) RGB*(quadcopter)+Z*(plane).

Examining a sampling of the tensors similar to those previously presented, Figures 3.13 to 3.17 show a remarkable dependence on the depth map, where the tensor samples seem to over-represent features recognizable from the single depth channel rather than features from the three color channels. Then, the depth data does contribute to the training independently from the color data, but in ways that cannot always be anticipated. For example, when the wrong depth map is paired with an image the network is not always capable of successfully classifying an object based off the RGB or Z information alone.

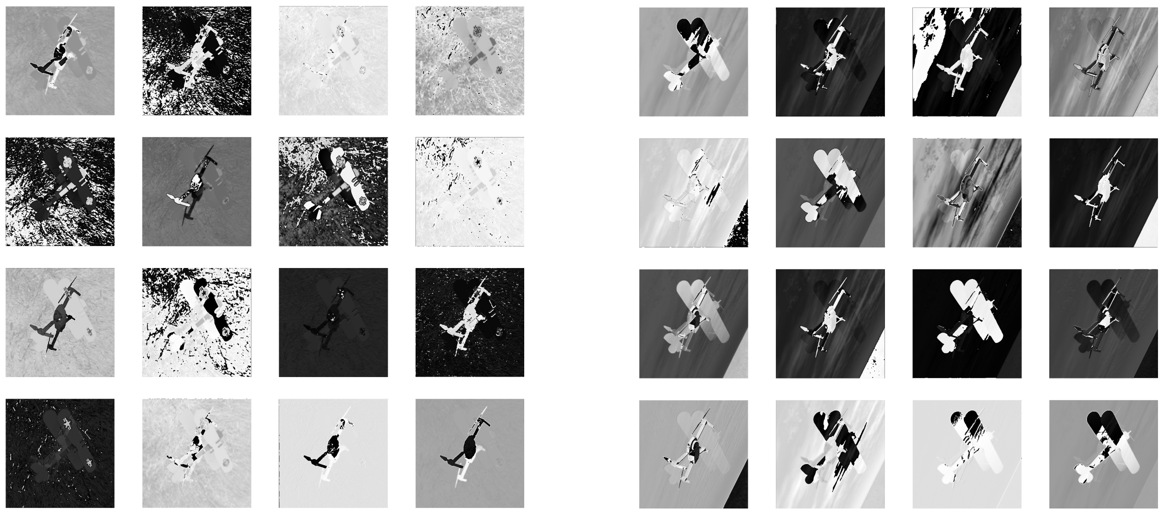


Fig. 3.13.: RGBZ* output of layer *conv1* in the ResNet-50 backbone showing the first sixteen 512x512 layers from the 512x512x64 tensor.
Left: RGB*(plane)+Z*(quadcopter). Right: RGB*(quadcopter)+Z*(plane).

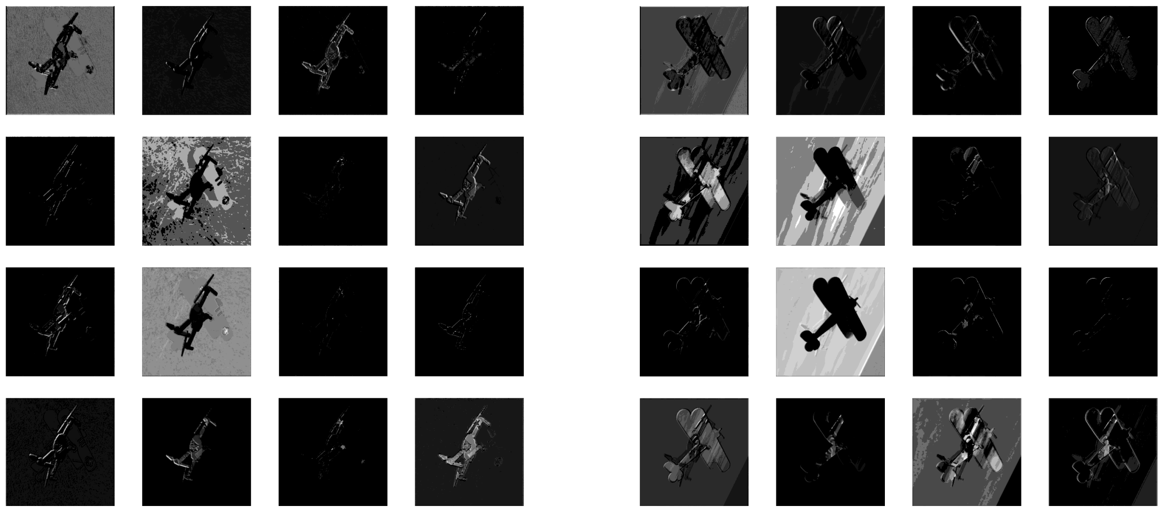


Fig. 3.14.: RGBZ* output of layer *res2c_out* in the ResNet-50 backbone showing the first sixteen 256x256 layers from the 256x256x256 tensor.
Left: RGB*(plane)+Z*(quadcopter). Right: RGB*(quadcopter)+Z*(plane).

The interdependence of RGB* and Z* data is shown in Figure 3.18, where the network returns a detection of "plane" with confidence score of 0.965 (out of 1) but a mask that is entirely dependent on the quadcopter shape drawn from the depth

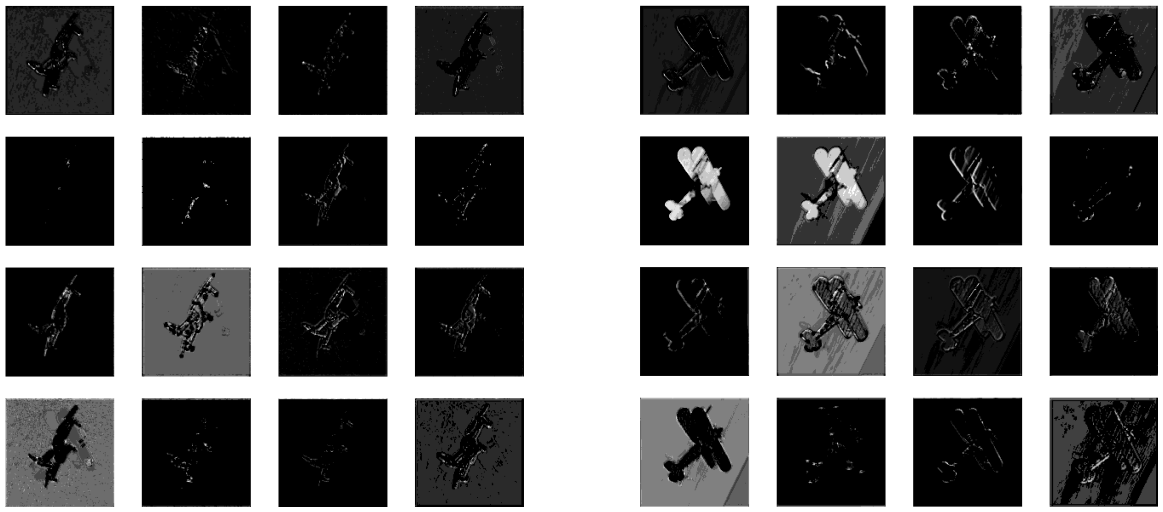


Fig. 3.15.: RGBZ* output of layer *res3d_out* in the ResNet-50 backbone showing the first sixteen 128x128 layers from the 128x128x512 tensor.
Left: RGB*(plane)+Z*(quadcopter). Right: RGB*(quadcopter)+Z*(plane).

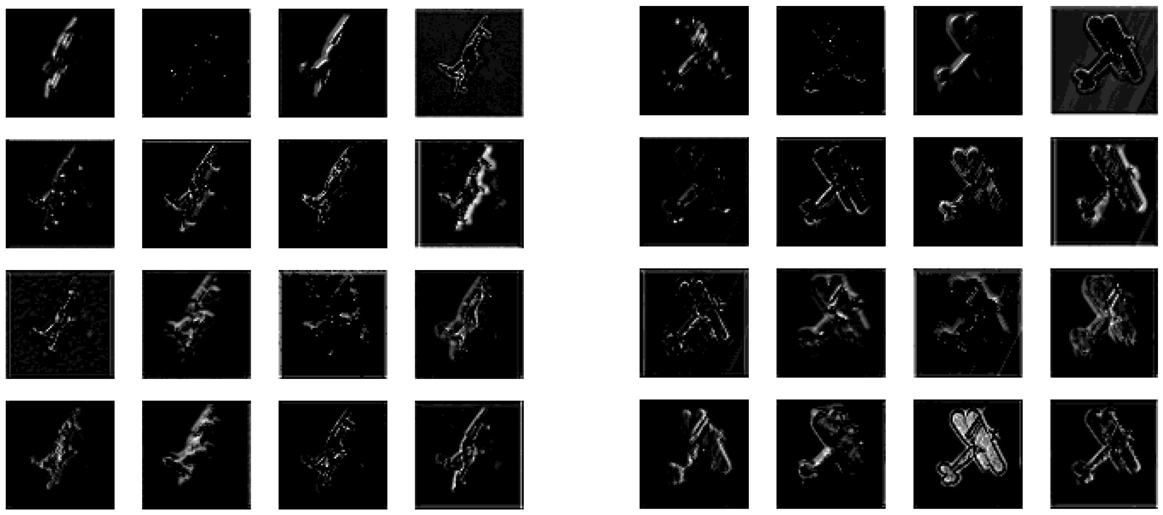


Fig. 3.16.: RGBZ* output of layer *res4c_out* in the ResNet-50 backbone showing the first sixteen 64x64 layers from the 64x64x1024 tensor.
Left: RGB*(plane)+Z*(quadcopter). Right: RGB*(quadcopter)+Z*(plane).

channel. In contrast, the complementary image of RGB*(quadcopter)+Z*(plane) in Figure 3.12(g) returned no detection, was strongly classified as background and out of the five classes had the highest confidence scores of 0.7 for the "eagle" class. When

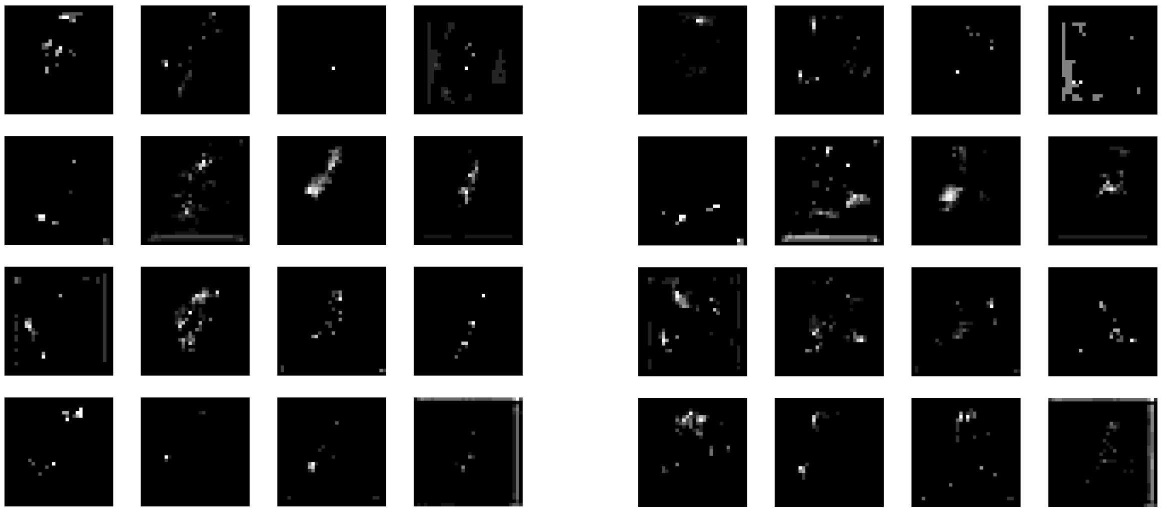


Fig. 3.17.: RGBZ* output of layer *res5c_out* in the ResNet-50 backbone showing the first sixteen 32x32 layers from the 32x32x2048 tensor.
Left: RGB*(plane)+Z*(quadcopter). Right: RGB*(quadcopter)+Z*(plane).

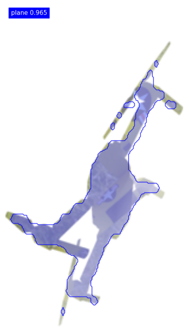


Fig. 3.18.: Network result on RGB*(plane)+Z*(quadcopter) image.

examining Figure 3.12(g), most of the color information surviving the depth channel masking effect is indeed background.

3.5 Conclusion

Depth information can be considered to act as gain added to the color signal, emphasizing important features and minimizing others. In cases like Figure 3.12(a) where the object to be detected is very close in frequency to the background, the depth information as structured for this work does an admirable job of isolating the object from the background and enabling a perfect detection and mask. In cases where the depth features could themselves be important for detection, e.g. the shape of the plane wings in Figure 3.12(f), avoiding any training examples where the depth channel is binarized is recommended. In the RGBZ+RGBZ* data set, such binarized depth maps represented at most 10% of all Z-depth training data, but it is possible that this was still a large enough percentage to prevent the network from consistently recognizing features only present in the depth data.

The depth data used for this work seemed to have the general effect of deleting features. When comparing the convolutional layer effects on the RGBZ* image tensors to the RGB* image tensors, features disappeared far earlier in the network than when using RGB* data. The disappearing features makes the ResNet-101 backbone a poor choice for this data, but without the extra convolutional layer features the testing performance suffers. The network continues to emphasize color-channel information for detection, but relies on edges for the mask generation. It is possible that a shallower CNN architecture such as ResNet37 could achieve performance gains similar to RGB* data with ResNet-101.

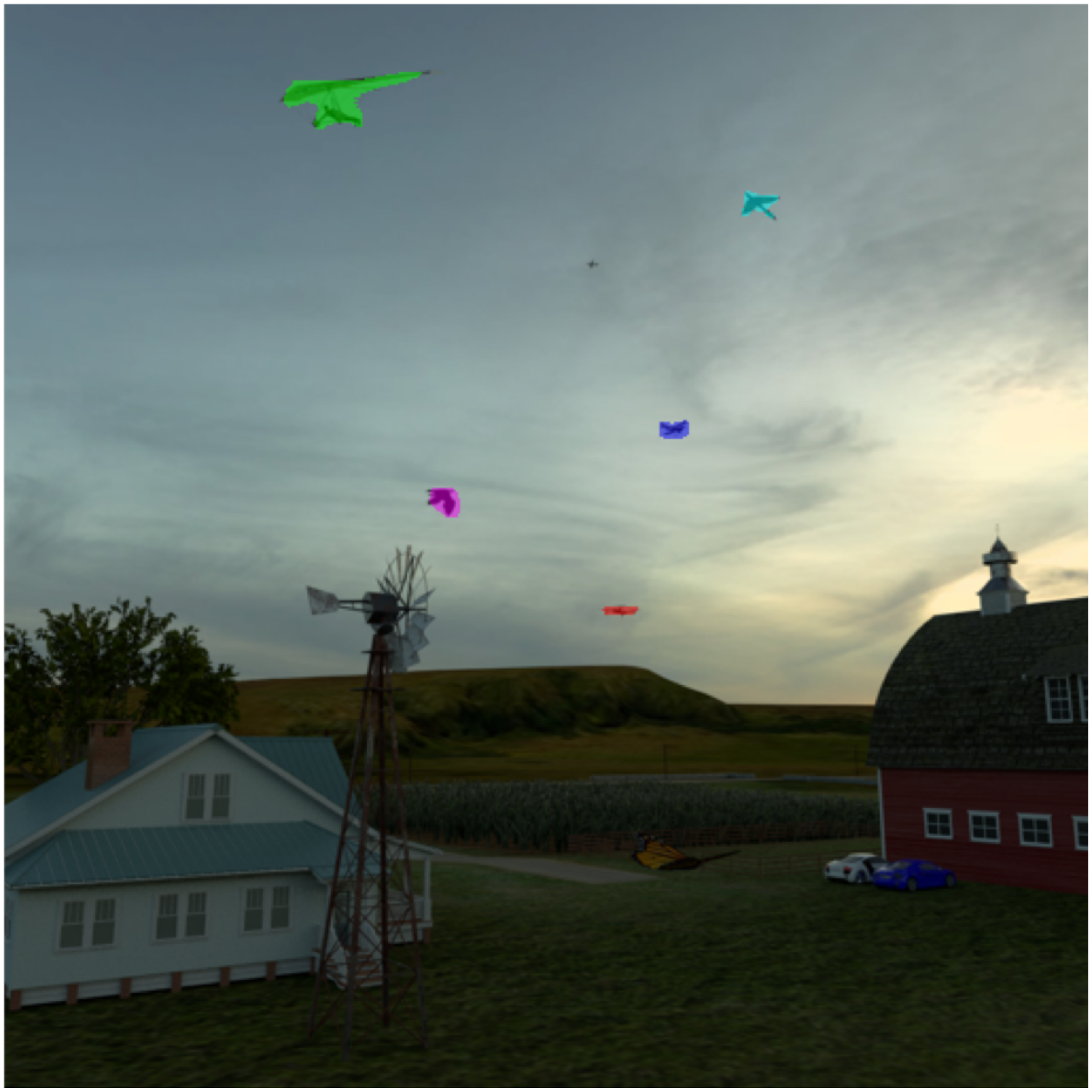


Fig. 3.19.: Results in RGBZ environment.

4. HOW REAL IS REAL? SIMILARITY OF VIRTUAL-WORLD DATA TO REAL-WORLD DATA

4.1 Introduction

In Chapters 2 and 3, syntheti virtual-world (VW) data was implemented in training with the assumption that the synthetic data shared an adequate number of features with real-world (RW) data to be useful. However, an analytical way of determining whether synthetic data is representative of RW data is desirable for two reasons: First, the effort to create the synthetic data set was by no means negligible, and all possible considerations should be taken to make certain that the efforts are not wasted in creating and/or using a synthetic data set with features not found in RW data. Second, an understanding of what features are most relevant to the training the network is useful when preparing both RW and synthetic data for network training; since implementing synthetic data provides the opportunity to precisely control every feature present in the data set, it is possible to more efficiently use RW data based on a developed understanding of the data set feature space.

By encoding data set features into latent variables and performing latent dimensionality analysis on each image in a data set, insight into the data set's internal structure becomes apparent. In the following sections, latent variables are generated for a portion of the RGB+RGB* data set using a Variational Autoencoder (VAE), then analyzed through Uniform Manifold Approximation and Projection (UMAP). Images generated by the VAE are additionally classified using an average performing network from Chapter 2.

4.2 Encoding of Features in Latent Space

Latent dimensional variables are used in statistical analysis as a dimensionality reduction technique to increase efficiency when building models for data. By defining a set of latent variables for RGB+RGB* data in a common latent space, it is possible to determine if RW and VW images share common features.

4.2.1 Data

The goal is not to analyze the entire RGB+RGB* data set feature space, but rather to determine if features in the RGB data are also present in the RGB* data, and if so, to what extent. We want to answer the question *Does a latent space embedding of RGB and RGB* data show a strong difference between RGB and RGB* data?* Accordingly, a subset of the complete RGB+RGB* data set was chosen, with details presented in Table 4.1. First, all images were downsampled to 128x128x3 pixels to make the data easily manipulable within the VAE architecture. The subset of the downsampled images was chosen such that objects with size less than 10x10 pixels were eliminated, the ratio of RGB to RGB* images increased, and image series with a large amount of repetition were removed. During training, 15% of the data was reserved as a validation data.

4.2.2 Variational Autoencoder Architecture

The Variational Autoencoder (VAE) architecture is presented in Figure 4.1. It consists of an Encoding network coupled to a Decoding network. In the Encoding network, the input layer for a 128x128x3 image which is reduced to a latent dimension vector of size 256. The Decoding network accepts the latent dimension vector as input, and computes an eigenimage for the Encoding network input. Training details are presented in Figure 4.2. The VAE was particularly chosen due to the use of

Table 4.1.: RGB+RGB* VAE Training Instances per Class

RGB+RGB* VAE Data		
	RGB Instances	RGB* Instances
Plane	93	405
Glider	97	711
Kite	97	303
Quadcopter	98	1038
Eagle	100	1424

a continuous Gaussian distribution, which allows for better interpolation between different classes [68].

Table 4.2.: VAE Network Training Parameters

VAE Training Details	
GPU	GE Force GTX 1080
MINIBATCH SIZE	8
IMAGE DIMENSION	128x128
LEARNING RATE	0.001
EPOCHS	5000
VALIDATION DATA	0.15

The network *epoch loss* was calculated as a weighted sum between the Kullback-Leibler (KL) loss and the Binary Cross Entropy (BCE) loss:

$$L = \{(Image\ Dimensions)^2 * (Binary\ Cross\ Entropy)\} + (KL\ Loss) \tag{4.1}$$

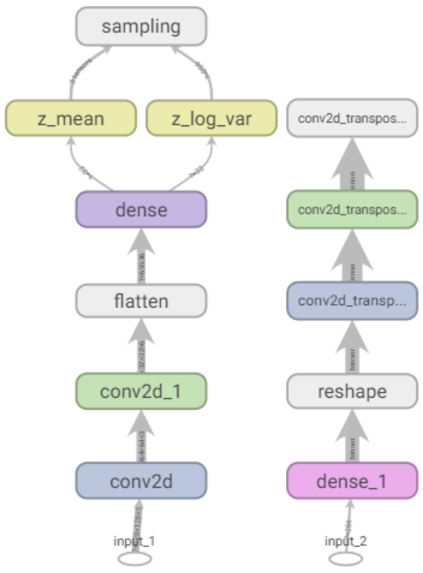


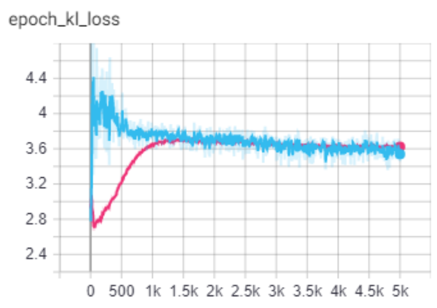
Fig. 4.1.: Architecture of the Variational Autoencoder.
Left: Encoder Sub-architecture. Right: Decoder Sub-architecture

where

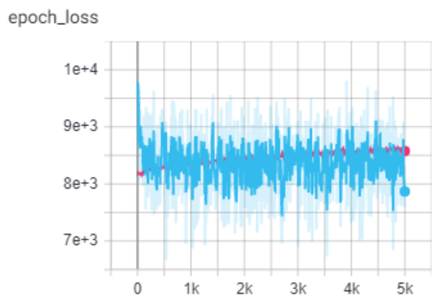
$$KL\ Loss = -\frac{1}{2}\{1 + z\_log\_var - z\_mean^2 - \exp z\_log\_var\} \tag{4.2}$$

.

In equation 4.2, the *z_log_var* and *z_mean* terms are MxN tensors output at the *z_mean* and *z_log_var* layers of the encoder as shown in Figure 4.1, where M is the number of images in the training set and N is the dimension of the latent space. Since the Binary Cross Entropy is weighted by 128^2, the epoch loss seeks to optimize image reconstruction with negligible contributions from the KL loss. This is clearly shown in Figure 4.2, where the epoch_kl_loss is on the order of 10^0, and the epoch_loss (containing the weighted BCE term) is on the order of 10^3.



(a)

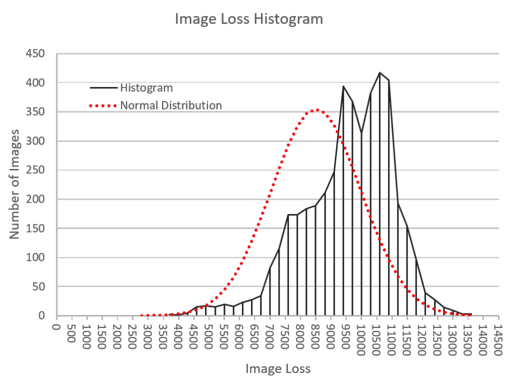


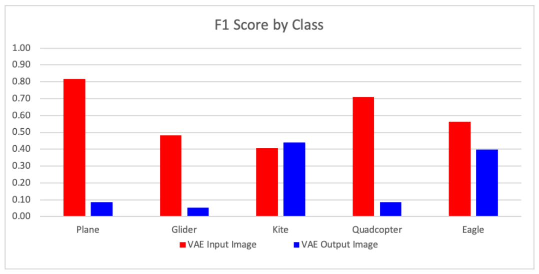
(b)

Fig. 4.2.: VAE training losses over 5000 epochs. Pink line is validation data; blue line is training data.
(a) KL Loss. (b) Reconstruction Loss with KL Loss.

4.2.3 Training Results

The distribution of reconstructed image losses are presented in Figure 4.3 (a) as a histogram with loss value per image binned in increments of 300. The normal distribution based on the mean image loss and variance in image loss values for the whole training set is imposed on the histogram for comparison. The goal is to move the entire distribution towards a lower image loss value. The original VAE input images and the eigenimages output by the VAE were classified using an MR-CNN trained with RGB+RGB* data to evaluate how similar the VAE data is to the original RGB+RGB* training data. The F1 scores are presented in Figure 4.3(b), and it is immediately obvious that the adjustments to the VAE input image set (downsampling and rescaling to 128x128) caused significant changes to the data set features, such that the VAE image set is no longer representative or the original RGB+RGB* image set. When the output eigenimages from the VAE are classified using the same RGB+RGB* MR-CNN, there is further decrease in F1 score in each of the five classes except for the ”kite” class. This further decrease in F1 score implies that the VAE latent space failed to capture most of the features important to MR-CNN.





(a) (b)

Fig. 4.3.: (a) Image Loss Histogram, showing distribution of loss scores per image. A normal distribution with the same mean and variance is fitted (dotted line) for comparison. (b) Comparison of F1 scores for VAE input images vs VAE output images using an average performing MR-CNN trained with RGB+RGB*.

Images with the lowest, mean, and highest loss scores that have been encoded and decoded by the VAE are presented in Figures 4.4 to 4.7. Additional images are presented in Figure 4.7 to demonstrate overall VAE performance. In general, every reconstructed image across the RW and VW data is similar to those presented here, with the exception of a portion of the RGB* eagle images and every RGB eagle image (which struggled in the VAE). Figure 4.8 contains images sampled directly from the latent space embeddings, where the first and last images in a row are eigenimages, and the intermediate images represent nine equally spaced increments in the 256-dimensional vector from the first eigenimage's vector to the second eigenimage's vector.

4.3 Analysis of Features in Latent Space

The VAE produced a latent variable embedding for each image as a byproduct of the image reconstruction process. These embeddings were analyzed using the Tensorboard Projection functionality to examine the data set's internal structure.

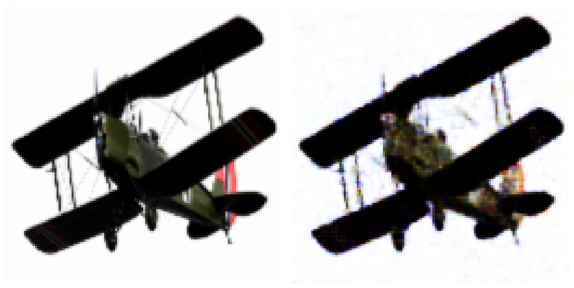


Fig. 4.4.: VAE best results.
Left: Original Image. [69] Right: Reconstructed Image with loss=2237.146

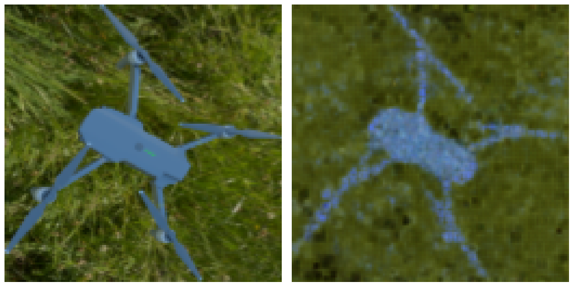


Fig. 4.5.: VAE average result.
Left: Original Image. Right: Reconstructed Image with loss=8509.311

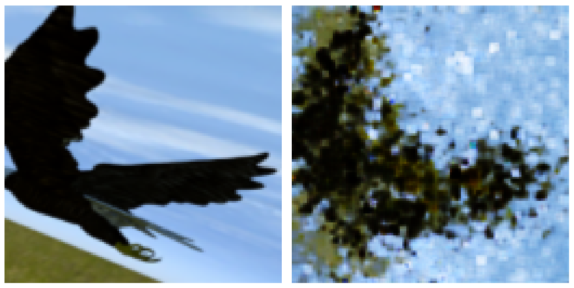


Fig. 4.6.: VAE worst result.
Left: Original Image. Right: Reconstructed Image with loss=12675.6

4.3.1 Methodology

Tensorboard automatically implements Principle Component Analysis (PCA) as a preliminary dimensionality reduction step to 200 dimensions (from 256). For the first time, the RGB+RGB* data set labels were adjusted to illustrate whether a point

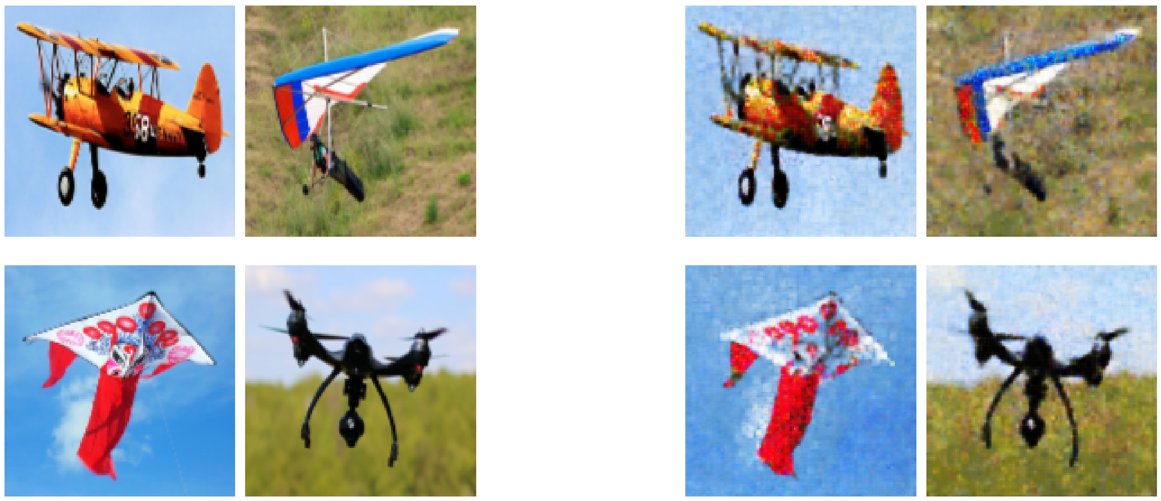


Fig. 4.7.: Examples of images encoded and decoded by the VAE.
Left: Original images: Biplane [70], Glider [71], Kite [72], Quadcopter [73]. Right: VAE reconstructions.

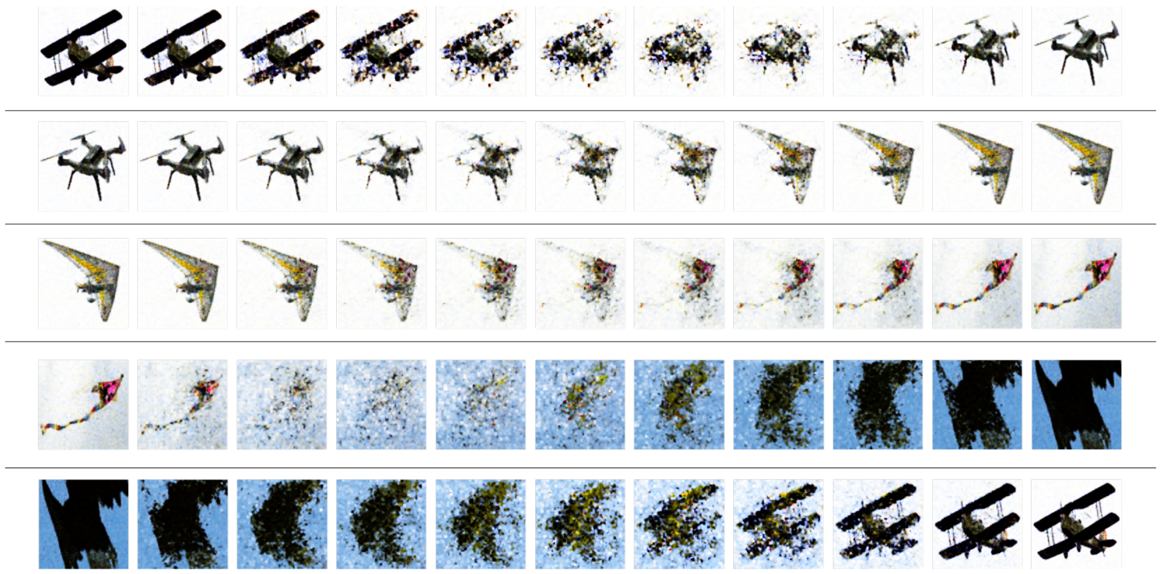


Fig. 4.8.: Images from the latent space. The first image and last images in a row are eigenimages. Intermediate images in row are generated from moving through the 256 dimensional latent space with equidistant steps from the first eigenimage vector to the second eigenimage vector.

belongs to RGB or RGB* and which of the five classes in RGB or RGB* (total 10 labels). Tensorboard automatically assigns colors to the class labels; these colors and the numeric Tensorboard encodings are shown in Figure 4.9, along with the PCA. The PCA shows a loose clustering by class, and an even looser bifurcation into two multi-class groups.

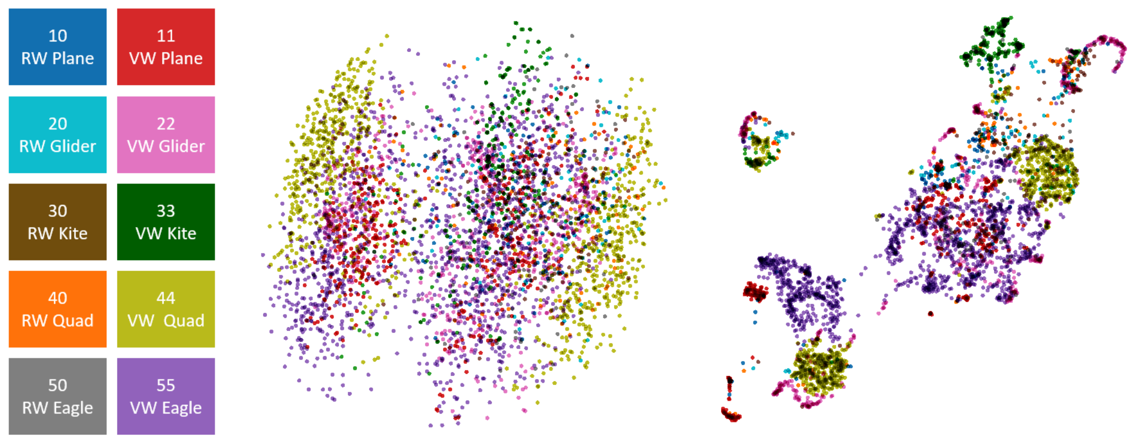


Fig. 4.9.: Latent space dimensionality reduction with PCA and UMAP. Left: Tensorboard color and label key. Middle: Tensorboard PCA of latent vectors. Right: Tensorboard UMAP of latent vectors.

Following the PCA operation, the Uniform Manifold Approximation and Projection (UMAP) was created using the Tensorboard functionality and 15 neighbors, and is also presented in Figure 4.9 on the far right. Unlike the PCA, the UMAP shows fairly tight clustering within classes, and it also shows a stronger clustering into several multi-class groups. This tight clustering is significant, in that (unlike PCA) the distance between two data points on the manifold projection has meaning [74].

4.3.2 Latent Space Results

To provide better understanding of how data was clustered, each class label was considered in terms of the high level class (“plane”, “glider”, “kite”, “quadcopter”, “eagle”) and the respective RW and VW subclasses (”RW plane”, ”VW plane”, etc.). The datapoints for these classes were highlighted in Tensorboard, and are presented in Figures 4.10 to 4.14.

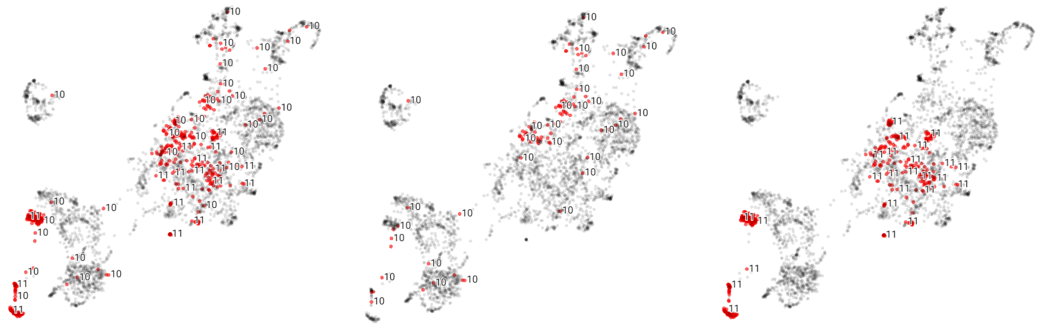


Fig. 4.10.: UMAP analysis showing "plane" labels.
Left: all labels. Middle: RW plane labels. Right: VW plane labels.

It immediately becomes apparent that the predominant feature causing clustering is something other than whether the datapoint in question is RW or VW. There is some variation in whether the RW points for a class are clustered adjacent to the VW points for a class, e.g. in Figure 4.14 the RW labels seem to be clustered predominantly with the VW labels, while in Figure 4.13 the RW labels seem to be dispersed away from the main VW clusterings.

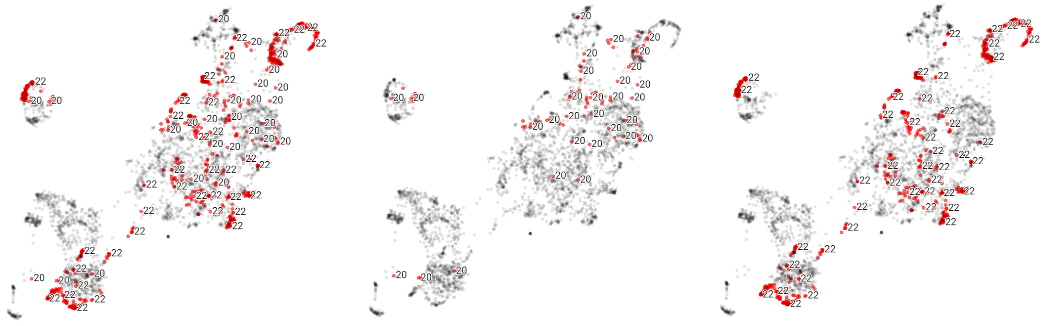


Fig. 4.11.: UMAP analysis showing "glider" labels.
Left: all labels. Middle: RW glider labels. Right: VW glider labels.

In each of the UMAP analysis figures, the VW data seems to be more tightly clustered than the RW data. In Figure 4.9, there are predominantly bright-green clusters corresponding to VW Quadcopter and light-purple clusters corresponding to

VW Eagle, but there are no corresponding tightly clustered RW groups. Instead, the RW image data for each class is more diffused across the manifold, implying that the shared features are found across class labels.

Special note should be taken of how not all the VW images for a single class are clustered together, either in the PCA or in the UMAP analysis. This can be partially explained by inspecting the sprite image for the VAE RGB+RGB* data set presented in Figure 4.15. A high-level glance immediately shows that the image backgrounds in the data set consist of largely green backgrounds (where the object is captured against the grass) and blue backgrounds (where the object is captured against the sky). This background dominance would explain why the split in VW data appeared beginning with the PCA in Figure 4.9 and persists through every class in the UMAP analysis. The finely checkered background in the sprite image belongs specifically to the VW kite class, and corresponds to the exclusively green "star" in the UMAP shown in Figure 4.9.

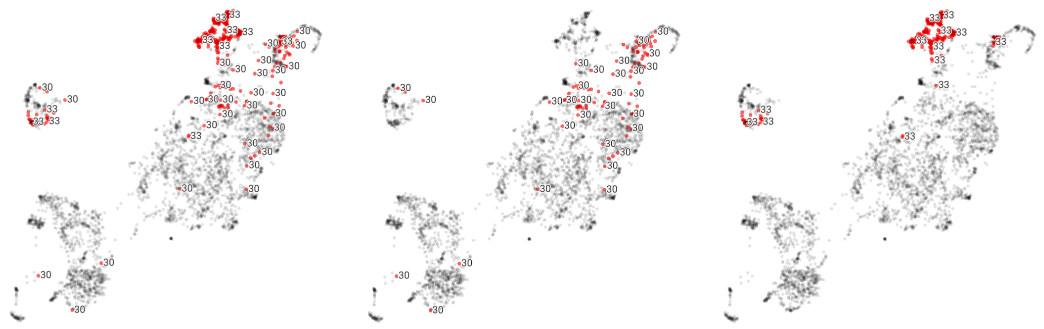


Fig. 4.12.: UMAP analysis showing "kite" labels.
Left: all labels. Middle: RW kite labels. Right: VW kite labels. Note that the "star" at the top of each UMAP corresponds to the finely-checkered kite background, and no RW images are present in this "star" cluster.

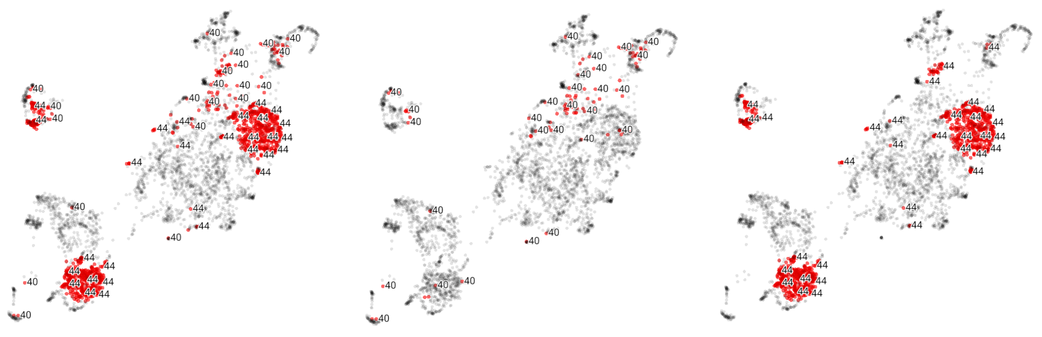


Fig. 4.13.: UMAP analysis showing “quadcopter” labels.
Left: all labels. Middle: RW quadcopter labels. Right: VW quadcopter labels.

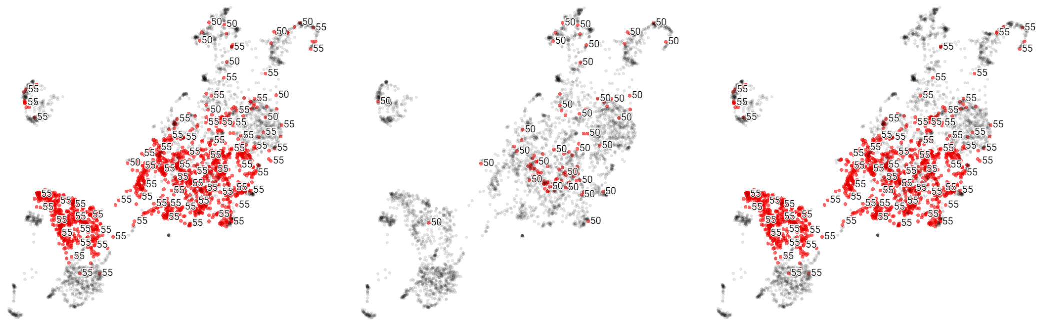


Fig. 4.14.: UMAP analysis showing “eagle” labels.
Left: all labels. MiddleL RW eagle labels. Right: VW eagle labels.

4.4 Conclusion

The analysis of latent dimension vectors generated by the VAE for the reduced RGB+RGB* indicates that the synthetic data does share features with the real-world data. Analysis of the UMAP distribution further indicates that color may be the most predominant feature shared by the RW and VW data. However, while MR-CNN largely ignores the background when making classifications, the UMAP analysis of the output VAE images seems to almost exclusively cluster based on background color. This implies that care should be taken that the color space mimics the RW image data when generating synthetic image data, and that a VAE structure that

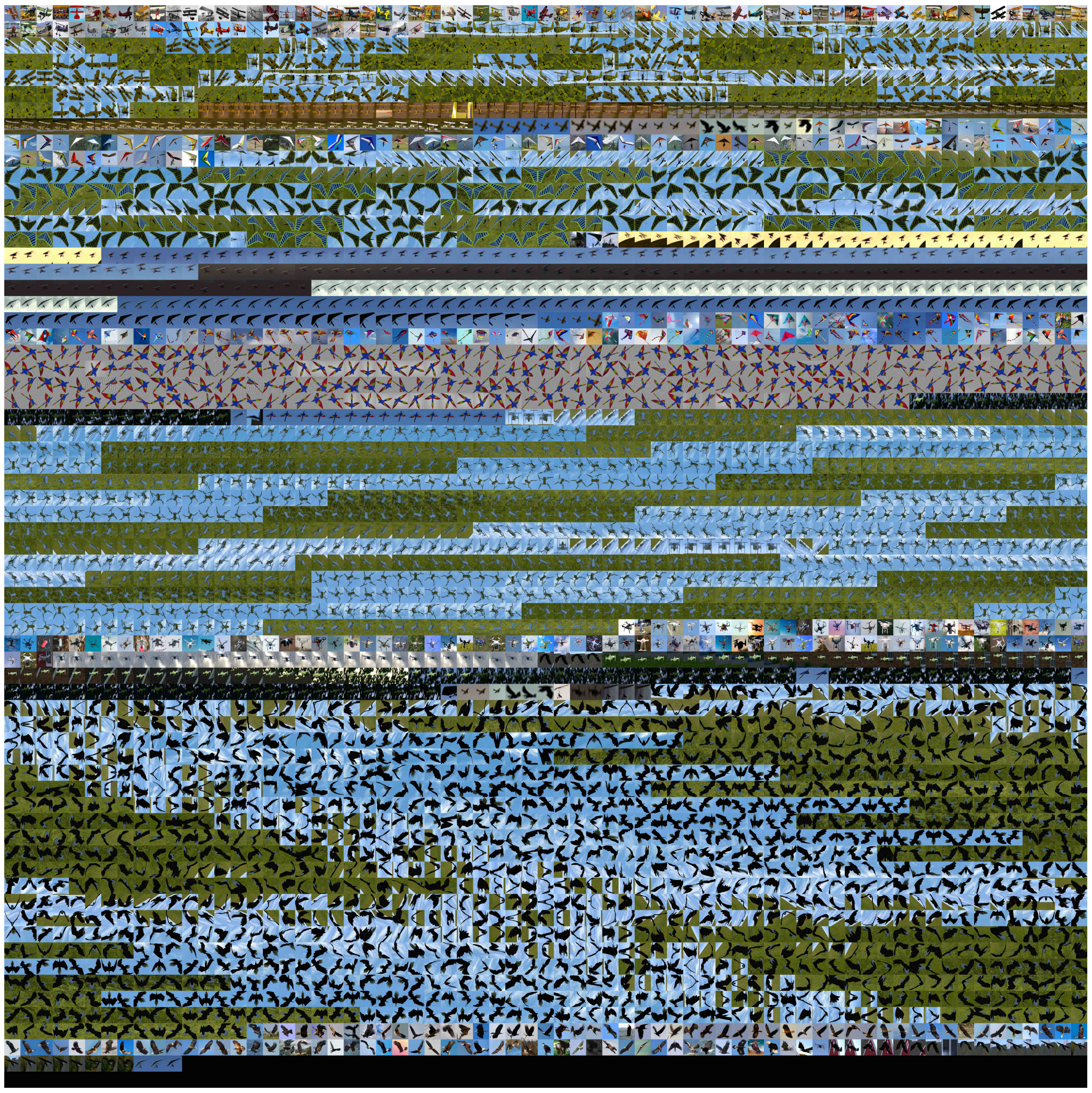


Fig. 4.15.: Sprite image showing entire VAE input data set.
There is a clear bias to blue and green backgrounds.

encourages the latent space to ignore background features may be helpful in further analyzing features relevant to the MR-CNN algorithm.

Continued training of the VAE may bring forward further features that distinguish between the RW and VW data, but care should be taken that the VAE does not minimize the differences between the RW and VW by minimizing the KL loss. In this work, the heavy weighting of the BCE loss made the contribution from the KL loss negligible. Future work should consider a loss based on recreating perceptual features such as a Gram matrix style loss, which allows context to play a role in feature determination.

5. SUMMARY

A synthetic data set consisting of five object classes in a variety of virtual environments and orientations was generated by a team of Media Arts and Sciences students. Denoted RGB(Z)*, this data set was paired with a smaller set of real-world RGB(Z) data and used to train the Mask R-CNN (MR-CNN) architecture in several configurations.

With depth data omitted, the RGB+RGB* data set was used with transfer learning implementing the MS COCO weights and the ResNet-101 backbone to improve the F1 scores of four out of the five object classes. The average maximum F1-score of all classes and all epochs for the networks trained with synthetic data is $F_1^* = 0.91$, compared to $F_1 = 0.89$ for the networks trained exclusively with real data, and the standard deviation of the maximum mean F1-score for synthetically trained networks is $\sigma_{F1}^* = 0.015$, compared to $\sigma_{F1} = 0.020$ for the networks trained exclusively with real data. The only class to not show improvement pre-existed in the MS-COCO classes, and had a baseline F1 score > 0.95. This implies that synthetic data has the capability to greatly improve network ability to learn and classify new data, especially where the F1 score is low or the class does not exist in the data used for transfer learning. Furthermore, the background of the RGB* data object was shown to have minimal impact on object classification, opening the door to the use of abstract backgrounds when realistic backgrounds are unavailable.

With depth data included, the RGBZ+RGBZ* data set was used with transfer learning implementing the ResNet-50 backbone in MR-CNN and relaxing the MS COCO weights throughout all the layers to allow the initial convolutional layers to train. By studying how the depth channel was convolved with the RGB channels, MR-CNN classifications were shown to primarily depend on the color features of an object, while the mask generated was shown to have high dependence on the depth channel. The importance of allowing the initial convolutional layer to train was shown by demonstrating the failure of the network to train when this layer was frozen with randomly initialized weights. Observations of decreased features in the ResNet-101 backbone when depth maps were included implies that shallower networks such as ResNet-34 may use depth data more effectively than deeper networks, despite the presence of additional features in a pre-trained ResNet-101 backbone.

Latent variables for a subset of the RGB+RGB* data set were generated using a Variational Autoencoder (VAE), then analyzed using Principle Component Analysis (PCA) and Uniform Manifold Approximation and Projection (UMAP). The latent variables showed no qualitative distinction between RGB and RGB* data, but did indicate a bias towards clustering based on image colorspace. This is consistent with the RGBZ* results from the MR-CNN architecture, where color determined classification when the RGB* channels were purposely mismatched with the Z* channel.

Future work should include intensive exploration of the specific depth features implemented during classification and mask generation for MR-CNN, and quantification as to the performance of shallower networks with more image channels. Real-world depth information should replace the hand-fabricated RW depth maps to allow for better analysis of the versimilitude of the synthetic depth data.

Additional explorations with the VAE are ongoing, including attempts to quantify the exact manifold generated and the distance between two data points in the original feature space rather than the latent dimesional space.

REFERENCES

REFERENCES

[1] N. Buch, S. A. Velastin, and J. Orwell, “A review of computer vision techniques for the analysis of urban traffic,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 920–939, 2011.

[2] Z. Wang, “The applications of deep learning on traffic identification,” *BlackHat USA*, vol. 24, no. 11, pp. 1–10, 2015.

[3] N. Kato, Z. M. Fadlullah, B. Mao, F. Tang, O. Akashi, T. Inoue, and K. Mizutani, “The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective,” *IEEE wireless communications*, vol. 24, no. 3, pp. 146–153, 2016.

[4] C.-J. Du and D.-W. Sun, “Learning techniques used in computer vision for food quality evaluation: a review,” *Journal of food engineering*, vol. 72, no. 1, pp. 39–55, 2006.

[5] T. Wang, Y. Chen, M. Qiao, and H. Snoussi, “A fast and robust convolutional neural network-based defect detection model in product quality control,” *The International Journal of Advanced Manufacturing Technology*, vol. 94, no. 9-12, pp. 3465–3471, 2018.

[6] D. Soukup and R. Huber-Mörk, “Convolutional neural networks for steel surface defect detection from photometric stereo images,” in *International Symposium on Visual Computing*. Springer, 2014, pp. 668–677.

[7] J. Yim, H. Jung, B. Yoo, C. Choi, D. Park, and J. Kim, “Rotating your face using multi-task deep neural network,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 676–684.

[8] B. Amos, B. Ludwiczuk, M. Satyanarayanan *et al.*, “Openface: A general-purpose face recognition library with mobile applications,” *CMU School of Computer Science*, vol. 6, p. 2, 2016.

[9] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, “A discriminative feature learning approach for deep face recognition,” in *European conference on computer vision*. Springer, 2016, pp. 499–515.

[10] D. C. Van Essen, C. H. Anderson, and D. J. Felleman, “Information processing in the primate visual system: an integrated systems perspective,” *Science*, vol. 255, no. 5043, pp. 419–423, 1992.

[11] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[12] T. Lindeberg, “Scale Invariant Feature Transform,” *Scholarpedia*, vol. 7, no. 5, p. 10491, 2012, revision #153939, last accessed 11/17/20.

[13] A. Kar *et al.*, "Skeletal tracking using microsoft kinect," *Methodology*, vol. 1, no. 1, p. 11, 2010.

[14] L.-G. Chen, C.-C. Cheng, Y.-M. Tsai, and L.-H. Huang, "3d depth generation by vanishing line detection," Apr. 1 2010, uS Patent App. 12/242,567.

[15] S. Hadfield, "Computer vision for the games industry," Ph.D. dissertation, University of Surrey Centre for Vision, Speech and Signal Processing, Guildford GU2 7XH United Kingdom, 2010.

[16] A. Saxena, J. Schulte, A. Y. Ng *et al.*, "Depth estimation using monocular and stereo cues." in *IJCAI*, vol. 7, 2007, pp. 2197–2203.

[17] L. Falkenhagen, "Depth estimation from stereoscopic image pairs assuming piecewise continuos surfaces," in *Image Processing for Broadcast and Video Production.* Springer, 1995, pp. 115–127.

[18] A. Rajagopalan, S. Chaudhuri, and U. Mudenagudi, "Depth estimation and image restoration using defocused stereo pairs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1521–1525, 2004.

[19] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.

[20] R. Garg, V. K. BG, G. Carneiro, and I. Reid, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," in *European Conference on Computer Vision.* Springer, 2016, pp. 740–756.

[21] I. Baldini, S. J. Fink, and E. Altman, "Predicting gpu performance from cpu runs using machine learning," in *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing.* IEEE, 2014, pp. 254–261.

[22] M. Jeon, S. Venkataraman, J. Qian, A. Phanishayee, W. Xiao, and F. Yang, "Multi-tenant gpu clusters for deep learning workloads: Analysis and implications," *Tech. Rep.*, 2018.

[23] T. Joachims, "Training linear svms in linear time," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 217–226.

[24] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *Journal of artificial intelligence research*, vol. 4, pp. 129–145, 1996.

[25] M. E. Taylor and P. Stone, "Cross-domain transfer for reinforcement learning," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 879–886.

[26] P. Wu and T. G. Dietterich, "Improving svm accuracy by training on auxiliary data sources," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 110.

[27] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017.

[28] A. Antoniou, A. Storkey, and H. Edwards, "Data augmentation generative adversarial networks," *arXiv:1711.04340*, 2018, last accessed 11/17/20.

[29] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[30] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu, "Boosting for transfer learning," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 193–200.

[31] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: Lessons learned from the 2015 mscoco image captioning challenge," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 4, pp. 652–663, 2016.

[32] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla, "Understanding real world indoor scenes with synthetic data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4077–4085.

[33] 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*, 2017, pp. 2107–2116.

[34] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European conference on computer vision*. Springer, 2016, pp. 102–118.

[35] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[36] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016.

[37] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

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

[39] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[40] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 886–893.

[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[42] 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*, 2016, pp. 770–778.

[43] S. Li, J. Jiao, Y. Han, and T. Weissman, “Demystifying resnet,” *arXiv:1611.01186*, 2017, last accessed 11/17/20.

[44] C. Lee, H. J. Kim, and K. W. Oh, “Comparison of faster r-cnn models for object detection,” in *2016 16th International Conference on Control, Automation and Systems (ICCAS).* IEEE, 2016, pp. 107–110.

[45] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[46] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[47] W. Abdulla, “Mask r-cnn for object detection and instance segmentation on keras and tensorflow,” https://github.com/matterport/Mask_RCNN, 2017, last accessed 11/17/20.

[48] R. Kasturi, D. Goldgof, P. Soundararajan, V. Manohar, J. Garofolo, R. Bowers, M. Boonstra, V. Korzhova, and J. Zhang, “Framework for performance evaluation of face, text, and vehicle detection and tracking in video: Data, metrics, and protocol,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 319–336, 2008.

[49] D. Hand and P. Christen, “A note on using the f-measure for evaluating record linkage algorithms,” *Statistics and Computing*, vol. 28, no. 3, pp. 539–547, 2018.

[50] M. Sokolova, N. Japkowicz, and S. Szpakowicz, “Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation,” in *Australasian joint conference on artificial intelligence.* Springer, 2006, pp. 1015–1021.

[51] D. M. W. Powers, “What the f-measure doesn’t measure: Features, flaws, fallacies and fixes,” *arXiv:1503.06410*, 2019, last accessed 11/17/20.

[52] J. Hoffmann, Y. Bar-Sinai, L. M. Lee, J. Andrejevic, S. Mishra, S. M. Rubinstein, and C. H. Rycroft, “Machine learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets,” *Science advances*, vol. 5, no. 4, p. eaau6792, 2019.

[53] E. A. Lopez-Rojas and S. Axelsson, “Money laundering detection using synthetic data,” in *The 27th annual workshop of the Swedish Artificial Intelligence Society (SAIS); 14-15 May 2012; Örebro; Sweden*, no. 071. Linköping University Electronic Press, 2012, pp. 33–40.

[54] D. Comaniciu, A. Kamen, D. Liu, B. Mailhe, and T. Mansi, “Image-based tumor phenotyping with machine learning from synthetic data,” May 7 2019, ’US Patent 10,282,588’.

[55] Q. Wang, J. Gao, W. Lin, and Y. Yuan, “Learning from synthetic data for crowd counting in the wild,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 8198–8207.

[56] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” *Lecture Notes in Computer Science*, p. 740–755, 2014, last accessed 11/17/20. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10602-1_48

[57] *Eagle*, February 2017, last accessed 11/17/20. [Online]. Available: https://pxhere.com/en/photo/805684

[58] jhmillard, *Plane*, September 2017, last accessed 11/17/20. [Online]. Available: https://pixabay.com/photos/plane-biplane-flying-old-plane-2765716/

[59] Sonia, *Hang Glider*, April 2014, last accessed 11/17/20. [Online]. Available: https://www.flickr.com/photos/sonialastrega/14003631634

[60] dotlizard, *Kite*, April 2015, last accessed 11/17/20. [Online]. Available: https://pixabay.com/photos/kite-flying-sky-dragon-wind-715478/

[61] *white quadcopter drone*, January 2017, last accessed 11/17/20. [Online]. Available: https://www.peakpx.com/309294/white-quadcopter-drone

[62] S. Song, S. P. Lichtenberg, and J. Xiao, “Sun rgb-d: A rgb-d scene understanding benchmark suite,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 567–576.

[63] K. Lai, L. Bo, X. Ren, and D. Fox, “A large-scale hierarchical multi-view rgb-d object dataset,” in *2011 IEEE international conference on robotics and automation.* IEEE, 2011, pp. 1817–1824.

[64] G. Pandey, J. R. McBride, and R. M. Eustice, “Ford campus vision and lidar data set,” *The International Journal of Robotics Research*, vol. 30, no. 13, pp. 1543–1552, 2011.

[65] Macgregor, *eagleTree*, July 2011, last accessed 12/01/20. [Online]. Available: https://flic.kr/p/apLVLC

[66] Z. Xu, S. Liu, J. Shi, and C. Lu, “Outdoor rgbd instance segmentation with residual regretting learning,” *IEEE Transactions on Image Processing*, vol. 29, pp. 5301–5309, 2020.

[67] M. Danielczuk, M. Matl, S. Gupta, A. Li, A. Lee, J. Mahler, and K. Goldberg, “Segmenting unknown 3d objects from real depth images using mask r-cnn trained on synthetic data,” in *2019 International Conference on Robotics and Automation (ICRA).* IEEE, 2019, pp. 7283–7290.

[68] B. Dai and D. Wipf, “Diagnosing and enhancing vae models,” *arXiv:1903.05789*, 2019, last accessed on 11/17/20.

[69] I. Wallmrod, *Biplane*, July 2016, last accessed 12/01/20. [Online]. Available: https:www.pexels.com/photo/green-and-black-propeller-plane-flying-under-clear-sky-126622/

[70] nightowl, *Biplane Airplane Oldtimer Yellow Fly Pilot Sky*, N/A N/A, last accessed 12/01/20. [Online]. Available: pixabay.com/photos/biplane-airplane-oldtimer-yellow-74554

[71] D. DeBold, *Hang Glider at Ed Levin Park*, June 2011, last accessed 12/01/20. [Online]. Available: https://www.flickr.com/photos/ddebold/5822443075

[72] Tholly, *Chinese Kite*, August 2006, last accessed 12/01/20. [Online]. Available: en.wikinews.org/wiki/User:Tholly

[73] N/A, *Quadcopter*, March 2017, last accessed 12/01/20. [Online]. Available: https://pxhere.com/en/photo/1180633

[74] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv:1802.03426*, 2020, last accessed on 11/17/20.