8-2019

On the Robustness of Object Detection Based Deep Learning Models

Matthew Seals
University of Tennessee

To the Graduate Council:

I am submitting herewith a thesis written by Matthew Seals entitled "On the Robustness of Object Detection Based Deep Learning Models." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Engineering.

Mongi Abidi, Major Professor

We have read this thesis and recommend its acceptance:

Hairong Qi, Qing Cao

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

On the Robustness of Object Detection Based Deep Learning Models

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Matthew Benjamin Seals

August 2019

Acknowledgements

First of all, I would like to thank everyone that has helped and supported me through this very long process. The first person I would like to thank is William Kuhn, the post-doctoral researcher who was working in the IRIS lab when I first started. Without his help, I would most likely still be working on completing the work for my thesis. He has helped me through this entire process, lending support when I needed it as well as much appreciated and helpful advice.

Second, I would like to thank my advisor, Dr. Abidi, for allowing me to perform research here in his lab. He has given me an abundance of helpful advice over the years as he was my undergraduate advisor too.

Next, I would like to thank my mother, Deena, and my father, Jerry, for their overwhelming support throughout the entire course of this thesis as well as throughout my life. I would have never been able to finish my master’s degree without both of their support. I also want to thank my sister, Paige, for her support as well.

I would like to thank my labmates I have had during the process, with the main ones being David Cornett and Clay Leach. They have both been great help in relieving some stressful situations with their banter.

The custom dataset used in this thesis contains data generated via the Zooniverse.org platform, development of which is funded by generous support, including a Global Impact Award from Google, and by a grant from the Alfred P. Sloan Foundation.

I would like to include special thanks to [4], [5], [28], [29], which I referred to many times while writing my thesis, for formatting and layout references. Also, I used [7] for basic CNN references and [8] for basic deep learning knowledge.

Abstract

Object detection is one of the most popular areas in the field of computer vision and deep learning. Several advances have been reported in the literature showing promising object detection results. However, most of these results use databases of images that have been collected under almost ideal conditions and tested with input images mostly not representative of real life imagery. When tested with challenging data, most of these object detection models break down.

The objective of this work is to quantify the performance of the most recent object detection models in the presence of realistic degradation in the form of differing levels of brightness, saturation, contrast, Gaussian blur, image size, sharpness, Gaussian noise, speckle noise, and salt and pepper noise. We have selected Faster RCNN as a typical model that is representative of the state of the art. We have used a binary class dataset from our laboratory for testing: Aphylla. We have also selected a popular multi-class dataset widely used by the community for our work: VOC2007.

We have conducted the following experiments (1) ran the model on the original pristine dataset and recorded the mAP score result, (2) ran the model on nine methods of degradation with 12 levels in each and recorded the mAP score results, and (3) compared the degradation results to one another to determine the model robustness. These experiments led to the clustering of the degradation models into three categories: high, medium, and low impact. These categories are based on the fluctuations within the results. The first class containing brightness and contrast resembles a Gaussian-like bell shaped curve with a plateau at the top. The second cluster contains Gaussian blur, image size, and all three types of noise resembles an exponential decay. The third category contains saturation and sharpness and has shown a small reduction in performance, which stays mostly uniform throughout the range.

The value of this research comes from studying the results and providing consistent guidance to the user as to which level of image degradation needs to be dealt with at a pre-processing stage to alleviate the drop in performance.

Table of Contents

List of Tables

List of Figures

1 Introduction

This research develops a framework for creating a trained object detection based deep learning model given a dataset and then determines the robustness of the trained model on degraded imagery. The first part of this thesis is centered around using a custom dataset to train the target neural network to obtain object detection results. The second portion of the thesis is centered around training the target network on a typical object detection dataset and then determining the model robustness using a testing set of degrading images.

In this thesis, we design a framework to be able to be interchangeable with different datasets, different deep learning neural networks, and different degradation models. We chose to test this framework on Faster RCNN using our custom dataset and the VOC2007 dataset with nine different degradation models separately. This allows us to prove this framework works for testing model robustness without having to be exhaustive in our work. This also allows our framework to be easily used in the future with different datasets, different CNNs, and other types of degradation models.

1.1 Motivation

The general goal for this research is to start by using a simple binary-class dataset for object detection, move onto a multi-class dataset, then, ultimately, perform a sensitivity analysis on the multi-class dataset. The overall motivation behind this is to determine the robustness and the impact of degraded images on object detection models. We start off using a binary-class object detection problem to see how well the model can perform given a simple problem. Then, we want to scale up the number of classes to produce more results. After we get multi-class working, we want to degrade the testing set of images and see how well the model from the last step performs on this non-pristine dataset. This will allow us to determine the robustness and impact of each method.

The motivation for the first section of this thesis – training a neural network on a binary-class dataset – comes from the previous work being done in the Imaging, Robotics, and Intelligent Systems (IRIS) laboratory by a post-doctoral researcher named William Kuhn. His doctorate research work was centered around taxonomy – the field of naming and classifying organisms [66]. His main focus inside this field is on the Odonata, which are dragonflies and damselflies. This leads into the creation of the Aphylla dataset, which he allowed us to use for this thesis. The objective he wanted to accomplish in his post-doctoral research in the IRIS labs was to be able to accurately detect Odonata within images taken in nature. With this task in mind, we decided to help him accomplish this by training a neural network using the Aphylla dataset. The end result would be a trained model which could detect the dragonflies within the given image. The future extensions of this work is to be able to count the number of dragonflies and then classify each detection.

For the second section of this thesis – testing the robustness of an object detection based deep learning model – the motivation comes from a question sparked by the first section of the work: given an image taken by any type of camera where the image is likely to be degraded in some form, how well would a deep learning model detect the objects in that image? This question gave us the motivation to create our easy to use framework to solve this problem. We decide to use a multi-class dataset to perform this sensitivity analysis because it will allow us to gather more results and, therefore, more conclusions. We assume (1) the deep learning model is trained on a mostly pristine, typical object detection dataset containing multiple classes and (2) the image contains a random type of degradation. To simulate the random type of degradation, we create our own degradation model to accomplish this task.

1.2 Pipeline

The process pipeline that we are presenting in this thesis is broken into two parts: a binary class framework and a multi-class framework, as shown and explained in Figure 1.1.

1.3 Contributions

The main contributions in this work are three-fold: (1) creating the framework/pipeline for determining model robustness, the whole of Figure 1.1; (2) the creation of the degradation model that is applied to the testing set of images before testing the model, the purple box in Figure 1.1; and (3) consistent guidelines and recommendations for mitigating the performance drops from the degradations. The first contribution will allow future extensions of this work to flow smoothly and produce robustness results with minimal effort. The second will allow any type of degradation to be applied onto the testing images for testing model robustness. The third will provide the user with guidelines about how to alleviate the performance drops caused by the degradation functions.

CONTRIBUTIONS:

1. Creation of the framework/pipeline
    a. Allows any dataset
    b. Allows any network
    c. Allows any model
2. Creation of the degradation model
    a. Nine degradation functions
    b. Allows any type of degradation
3. Consistent guidelines and recommendations for mitigating performance drops

The pipeline we created is the overarching contribution within this thesis. This includes the degradation model we created. This pipeline lays down the groundwork needed for testing model robustness with any base network and any object detection model available, whether that be in past, present, or future networks and models. This allows this thesis work to be used by anyone wanting to test for any type of model robustness on any kind of degradation techniques. In this

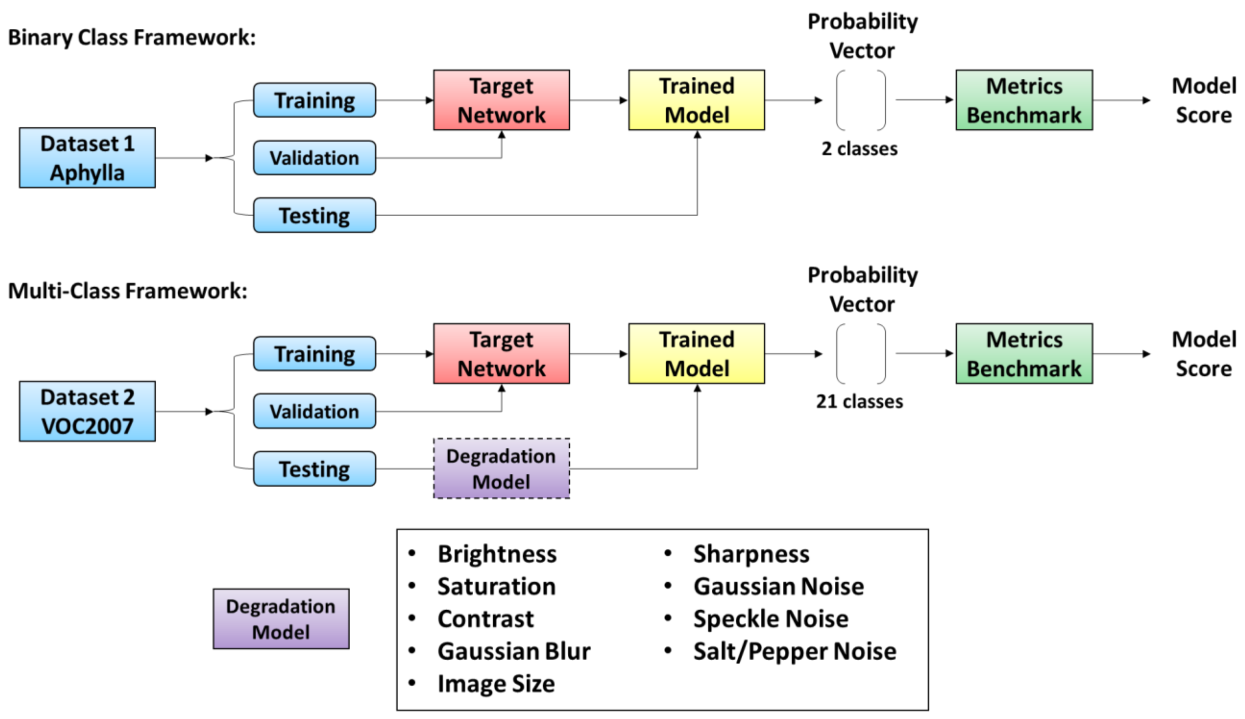


Figure 1.1. Both model frameworks used within this research work. *The first framework is for binary class object detection, meaning a single object and the background. The second is for multi-class object detection, which uses the VOC2007 dataset that contains 20 objects and the background. Both frameworks follow the same format. First, we split the dataset into training, validation, and testing sets. We, then, feed the training set into the target network, and we use the validation set to validate the current training iteration once it finishes to determine if the network should re-train on the training set. Once this process is finished, we have a trained model. We feed the testing set into the trained model to obtain a probability vector containing a probability for each detection. Then, we use the probabilities to determine the model score via using a benchmarking metric, the mean average precision score. The addition we include in the second framework is the degradation model that we created. We have, in total, nine different degradation models, which are all listed within the figure.*

thesis, we choose to demonstrate our pipeline using our binary class, custom dataset – Aphylla – as well as using the multi-class PASCAL VOC2007 dataset and using the VGG16 base network with the Faster RCNN object detection model. These are examples of the pipeline being used, even though any network and model can be used.

The degradation model we created sits in the pipeline between the testing set of images and the trained model and is our second major contribution. This model allows for any type of degradation to be applied to the testing images, and conversely, to the training and validation sets of images too. This will allow future works to be able to use our degradation model to test for many different types of model robustness beyond what we test in this research. This thesis applies 12 levels of nine different degradation types to the test set of images. We chose these nine types to demonstrate how our model integrates in the pipeline and show how simple it is to take one of the degradation techniques and change it to another.

The consistent guidelines and recommendations for mitigating performance drops we provide is our third major contribution. The three categories we classify each degradation type into are high, medium, and low impact. These categories are based on the fluctuations within the results. The high category resembles a Gaussian-like bell shaped curve that has a plateau at the top. This plateau covers a decent portion in both the brightness and contrast graphs, from 50% to 200%. Outside of this plateau, the fringes of the graph resemble exponential decays. An example of the guideline we provide based on these characteristics is at which point in the range would a pre-processing step need to be performed on the images before object recognition can be performed in order to minimize the performance decrease. For instance, in the second category, the results resemble mild to severe exponential decays throughout the entire range. In order to delay the model performance effects of these types of degradation, a pre-processing step, such as median filtering, needs to be applied to the test images. For the third class, the results show a small reduction in object detection performance, which stays mostly uniform throughout the range. This means that no action is needed.

1.4 Synopsis

In Chapter 2, we discuss our custom dataset as well as study the previous work completed in this field related to public datasets, deep learning models, and sensitivity analysis. In Chapter 3, we discuss the methodology of training the network on the custom dataset as well as the process of determining model robustness on degraded imagery. In Chapter 4, we discuss our results obtained from both sections of our work and include recommendations that need to be taken based on the model performance on each degradation technique. Lastly, in Chapter 5, we conclude the thesis with the summary of our work as well as possibilities of future work.

2 Literature Review

This chapter will consist of descriptions and background of the datasets used in this thesis, the history of CNNs, the background works related to object detection using deep learning networks, and related works pertaining to performing sensitivity analyses. The ordering of this chapter is based on the pipeline we created in Figure 1.1. The pipeline (and ordering of this chapter) starts with the datasets being used by the CNN. After the CNN is fully trained by the datasets, the object detection model uses the trained network, which produces the detection results. These results are used to perform the sensitivity analysis. We set this chapter up this way to help illustrate how and in what order the pipeline is being used.

2.1 Datasets

In the recent years, object recognition and detection tasks have become particularly popular, and these require large annotated image datasets to train. If the dataset is not sufficient in size, the model can quickly overfit the data, meaning it cannot be generalized to any new images since it is overly fine-tuned to the current images [53]. The datasets also must be annotated, which is an exceptionally tedious task because it requires a large amount of manual labor since the object outlines or bounding boxes have to be accurately created by humans. These two requirements, being large-scale and annotated, led to organizations creating privately owned only for in-house use as well as publicly available image datasets so there would be standardized ways to train object detection models. For the privately owned datasets, researchers were creating them for their own projects and would only publish the datasets if their work was published to a high profile journal [34], [45]. We have created our own private dataset as well. It is aptly named Aphylla after a genus of dragonfly since the dataset consists of only dragonflies. For publicly owned datasets, three main image datasets, which are also annotated, have been made specifically for object detection tasks: PASCAL VOC (pattern analysis, statistical modeling, and computational learning visual object classes) [46], ImageNet [25], and COCO (Common Objects in Context) [33]. The CIFAR-10 and CIFAR-100 datasets are publicly available as well [6], [27]. In this research, we use our dataset – Aphylla – as well as both the PASCAL VOC2007 and ImageNet datasets separately to train our model.

2.1.1 Binary Class Dataset – Aphylla

The dataset, compiled by William Kuhn, is named Aphylla after a genus of dragonfly and consists of 110,750 images containing at least one dragonfly or damselfly per image. A few example images are shown in Figure 2.1. All of the images in the dataset were copied with permission from OdonataCentral (OC) [58]. A portion of the dataset consists of 14,740 annotated images, which include all four corners of the bounding box that encompasses each dragonfly and damselfly. The annotations were created in the Zen of Dragons project on the crowdsourcing platform Zooniverse.org [59], where users drew a tight bounding box around each dragonfly and damselfly

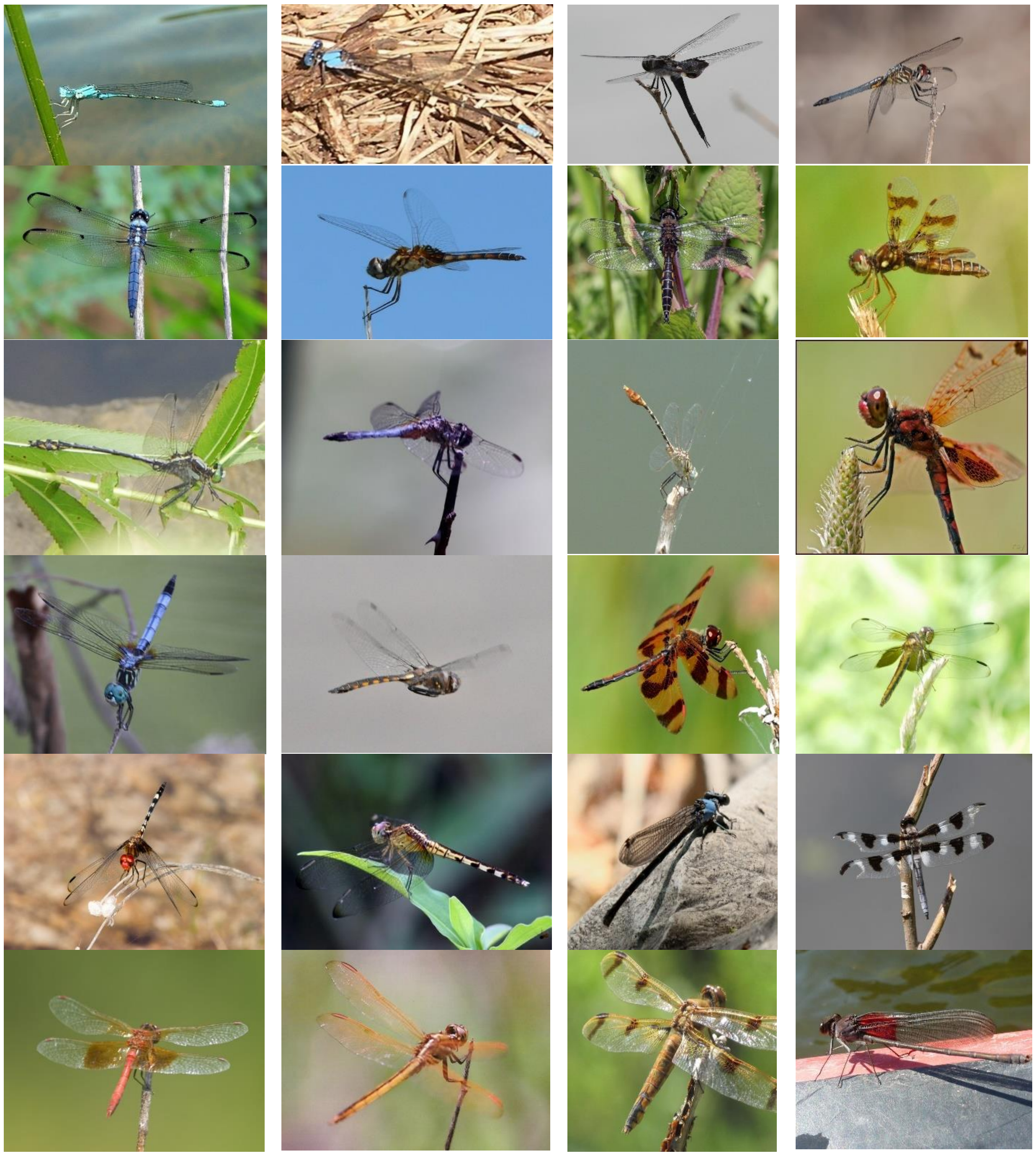


Figure 2.1 Example images of the Aphylla dataset. *A few example images from the Aphylla dataset. These images show close ups of dragonflies, which is what this dataset mainly consists of. It also contains many images showing multiple dragonflies in a single image.*

in each of the 14,740 images. To make sure the annotations were accurate, each image was annotated by four users, and the coordinates of the bounding boxes represent the average of the two closest ones for each object. We use the annotated portion of the dataset to train our model.

2.1.2 Multi-Class Datasets – PASCAL VOC and ImageNet

We use the PASCAL VOC2007 and ImageNet datasets for training our model, so these are the only two groups of datasets we will discuss in this section. There are a vast amount of other publicly available, multi-class datasets used within the field of deep learning that we chose not to discuss in this section for clarity.

The PASCAL VOC and ImageNet datasets are multi-class and publicly available for anyone to use for research purposes. The model we use in our research is pre-trained with the ImageNet dataset and fine-tuned on the VOC2007 dataset, which is one of the many PASCAL VOC datasets. We chose to demonstrate the research work using the VOC2007 dataset to train the network even though any dataset can easily be used. We wanted to narrow down the work for the sake of time. The pipeline in Figure 1.1 makes changing the dataset in the future trivial.

The following sub-sections are going to talk about the descriptions and background work behind the PASCAL VOC datasets – specifically VOC2007, the one used in the thesis – and the ImageNet dataset – the one used to pre-train our network.

2.1.2.1 PASCAL VOC

The PASCAL VOC datasets are some of the most widely used public datasets for object detection. A dataset was created for the PASCAL VOC challenge every year from 2005 to 2012, but the VOC2007 and the VOC2012 datasets are the most popular, e.g. [18], [44], [50], and more make use of these two specific datasets. The 2007 dataset encompasses 20 classes of objects (defined later), which has been a fixed number ever since [46]. All VOC datasets consist of the images with bounding box annotations, and the 2008 dataset and later also have segmentation annotations. The bounding box annotations include the coordinates for the four corners of the box in which the object is tightly encompassed as well as the class of the object. In contrast, the segmentation annotations include which object class each individual pixel is. The images and the annotations in the VOC datasets are broken into training, validation, and testing sets before they are made publicly available. Although the split can be arbitrary, the split used for the VOC dataset is 25%/25%/50% (training/validation/testing), meaning 25% of the data is used to train the model, etc. [46]. The substantial portion of images in the testing set are used for rigorous evaluations of the models submitted to the PASCAL VOC challenge [46]. As of 2008, testing annotations were no longer released to prevent model fabrication via parameter tuning – creating a model to specifically perform well on the testing data and nothing else [46]. We chose to use the VOC2007 dataset in this research as the testing annotations were needed to test our model without having to reduce the training and validation set sizes.

Each year, the VOC datasets were created by using images from the photo-sharing website Flickr (flickr.com), which are consumer images [46]. The search terms used in creating the datasets are exceedingly general, i.e. they include the "targeted" class name, synonyms, and scenes or situations where the class is likely to occur [46]. An example would be the class "cow" where they also use the terms beef, heifer, moo, dairy, milk, milking, and farm in the Flickr search [46]. This resulted in a seemingly unbiased dataset because the images were randomly selected from the site rather than an image scientist creating or selecting the images for the dataset with the sole purpose of object detection/recognition. Since they were randomly selected, the images consist of a variety of object sizes, orientations, poses/angles, and illumination [46]. The images also contain a variation of object placement or position, for example, there are images with dining tables and chairs in a room scene containing other objects, rather than the dining table and chairs being the focus of the image. A few of the images are shown in Figure 2.2.

After the VOC2007 dataset was created, a small bias was discovered in the creation of the dataset due to the way the images were collected. Flickr used (and may still use as default) a method called "recency" when it returned search results, which means the most recent images are returned first [46]. The VOC datasets were created by generating 100,000 images (100 pages of 1,000 images each) for each search term and selecting a random image from each page, and since the VOC2007 dataset was created in January of 2007, it led to an abnormal number of Christmas/winter-related images being included. For example, the dataset contains a larger number of Christmas trees than it would have contained at any other time of the year. This was alleviated in the later years by including a randomized date in each search term, which would return a more normalized set of images [46].

Once the images were selected for the dataset, there were strict guidelines set in place for the creation of the annotations by Everingham et al., the creators of the dataset, in [46]. These guidelines lay out all of the specifics of the objects being annotated, such as what to label, image quality, bounding box details, accuracy, etc. The guidelines also break down the specifics of each class and what not to include, e.g. do not include lions in the "cat" class or benches/stools in the "chair" class. This was necessary to maintain consistency and accuracy throughout the entire dataset, which would guarantee accurate training and evaluation of models trained on this dataset.

The dataset is divided into two main subsets: training/validation set (trainval) and test set, with the trainval set further divided into suggested training (train) and validation (val) sets [46]. In total, the dataset contains 9,963 images consisting of 24,640 annotated objects and are split with 50% in the trainval set (train: 2501; val: 2510) and 50% in the test set (test: 4952). These are already split beforehand, but the users are also able to combine, randomize, and split the images again as they see fit as long as the image sets contain a relatively equal number of images from each class in both the trainval and test sets. If the sets are not relatively equal in this regard, it could potentially lead to a biased model – able to classify certain objects really well but performs poorly on the rest. The splits as they are released is also how the submitted models into the challenge will be evaluated.

The statistics for how the 20 classes in the VOC2007 dataset are broken down is as such: aeroplane (238 trainval images, 204 test images); bicycle (243, 239); bird (330, 282); boat (181, 172); bottle

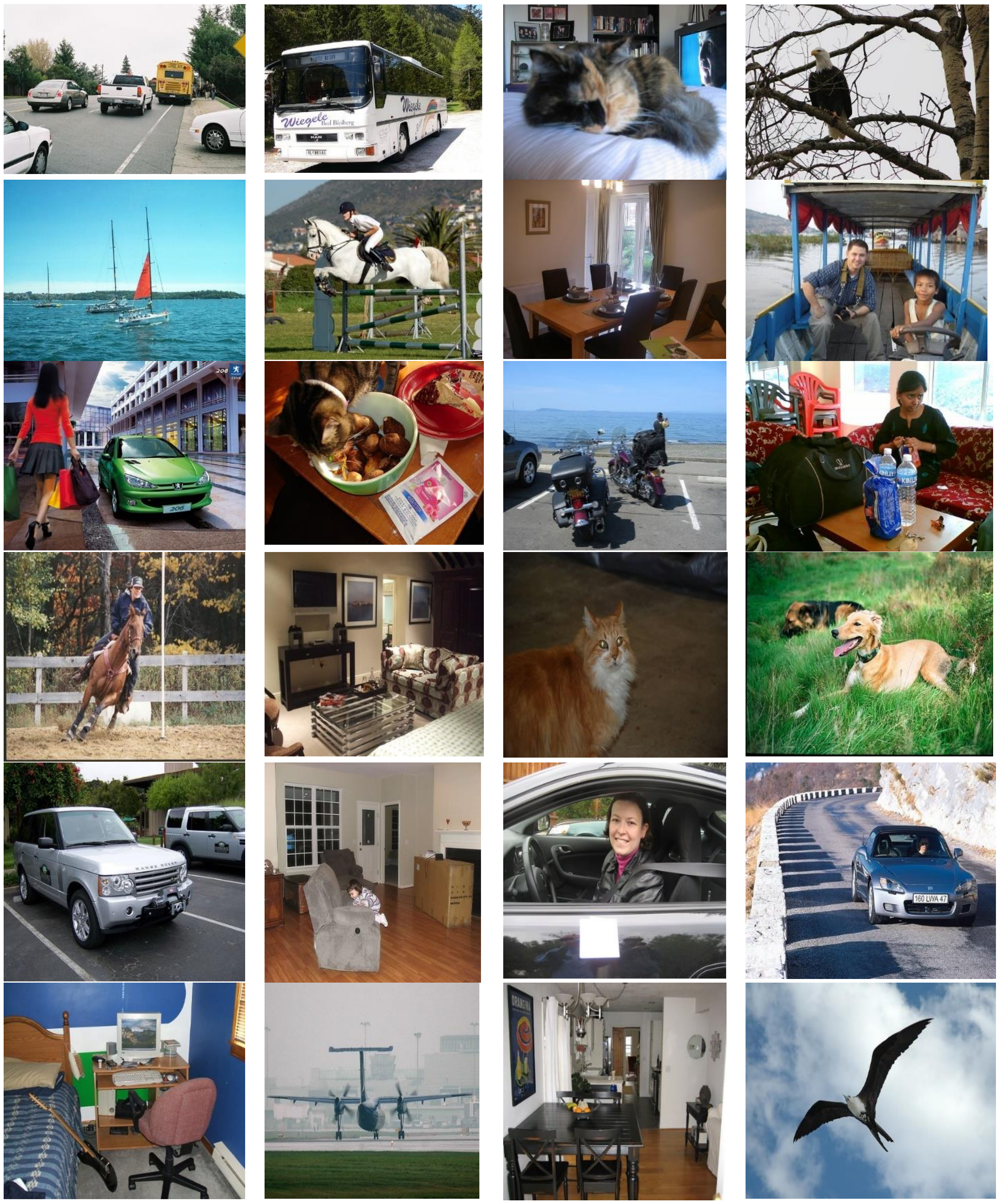


Figure 2.2 Example images from the VOC2007 dataset. *A few example images from the VOC2007 dataset showing cars, buses, boats, people, birds, dining table and chairs, horses, cats, and more.*

(244, 212); bus (186, 174); car (713, 721); cat (337, 322); chair (445, 417); cow (141, 127); dining table (200, 190); dog (421, 418); horse (287, 274); motorbike (245, 222); person (2008, 2007); potted plant (245, 224); sheep (96, 97); sofa (229, 223); train (261, 259); and tv/monitor (256, 229), adapted from Table 2 in [46]. The numbers in the trainval set versus the test set for all objects are as close in size to one another as the creators could get given there are multiple objects per image.

2.1.2.2 ImageNet Description

At over 15 million images with over 22,000 categories, ImageNet is the largest, public image dataset used for object recognition and detection research [25]. WordNet is used as the backbone of ImageNet, meaning the words in WordNet are first disambiguated, then the synonyms are combined together to create the object categories. ImageNet consists of all third party imagery that has been annotated using Amazon Mechanical Turk, or MTurk, where manual tasks can be posted for a small monetary reward [25], [57]. The ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been hosted every year since 2010 using a subset of the entire ImageNet database, which contains roughly 1.2 million images evenly broken down into 1,000 non-overlapping categories [25].

2.1.2.3 Dataset Summary

In the previous sub-sections, we discuss the PASCAL VOC datasets as well as the ImageNet dataset. Within the thesis work, we specifically used the VOC2007 dataset to fine-tune the network and the ImageNet dataset as the pre-trained network. There are many other datasets not covered here because we chose to use these two datasets to prove that the aforementioned pipeline (shown in Figure 1.1) works.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a stepping stone into the field of deep learning, which is a category of machine learning that is used specifically for feature extraction. The extracted features can be acoustic models for speech recognition [10], [42], lexical items for natural language processing [2], visual cues for translating videos to natural sentences [47], or image features for image processing (among other tasks). In [31], [32], deep neural networks learned to master the game of Go using supervised learning from human expert games and reinforcement learning from games of self-play, called AlphaGo. This is only one of the many examples there are of deep neural networks achieving remarkable results. Arguably, the most popular application of CNNs is image processing.

This section covers the history of CNNs starting off with the object recognition problems CNNs have faced, the structure of CNNs, and the many different networks covered in chronological order. We ultimately decided on using the VGGNet as our base model and Faster RCNN as our object detection model.

2.2.1 *Object Recognition and Detection Problems*

CNN architectures that have been created for object recognition and detection are explicitly built with images as inputs, which allow certain assumptions to be made to make the CNNs more efficient by reducing the number of parameters. Traditional neural networks only allow the neurons to react to a single point in the input whereas convolutional neural networks allow the neurons to react to a region of the input. This region is determined by the convolutional kernel size and allows the CNN to learn object features from images. When CNNs were first introduced in 1998 [21], the major drawback was the vast amount of computational power required to apply them to anything larger than 32x32x3 images. Today, that is not an issue, especially with how large GPU memory space has gotten due to Moore's Law, which allows the training data to be much higher resolution imagery. Training data has grown in size significantly since then as well. These two reasons combined can help explain why CNNs have become immensely more popular since 1998.

2.2.2 *Structure of Convolutional Neural Networks*

The typical structure of a CNN is as follows:

$$\mathrm{INPUT} \rightarrow \mathrm{CONV} \rightarrow \mathrm{ReLU} \rightarrow \mathrm{POOL} \rightarrow \mathrm{FC} \rightarrow \mathrm{OUT} \qquad (2.1)$$

where INPUT is the input images; CONV stands for the convolutional layers within the architecture; ReLU means rectified linear unit, which is placed after each CONV; POOL is the pooling layer after each ReLU; FC stands for the fully connected layers used for classification at the end of the pipeline; and OUT is the output feature maps. The input image is fed into the network as an array, and the first layer is always a convolutional layer[1]. This layer uses the convolutional kernel to create the next layer of hidden neurons. The kernel determines the region of pixels in the input layer that will be connected to each neuron in the hidden layer, which is called the receptive field, shown in Figure 2.3. Almost all CNN networks contain a varying number of convolutional layers as well as varying sizes of the convolutional kernels within each layer.

The next step in the CNN pipeline is the activation function, which is applied to the resulting hidden layer. One of the most popular activation functions, ReLU, is defined as

$$f(x) = \max(0, x) \qquad (2.2)$$

[1] There have been recent works using genetic algorithms to determine the best structure for the layers of a Deep CNN [19], [12].

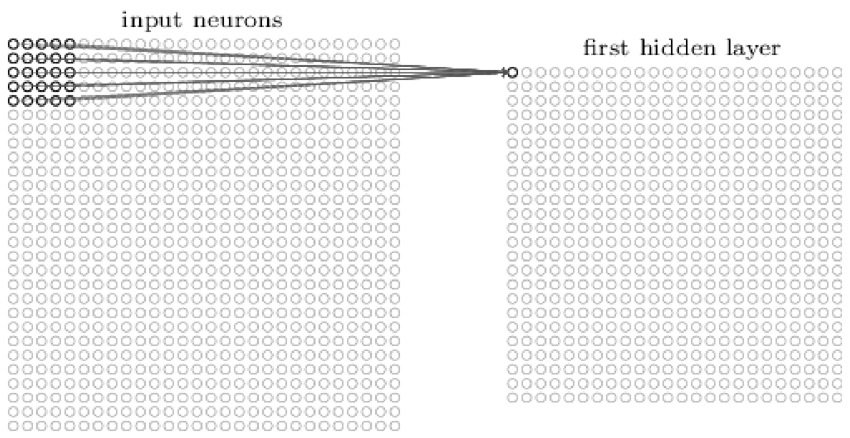


Figure 2.3 Example of a 5x5 convolutional kernel being applied to a 28x28x1 image, *from [54]. The convolutional kernel is used in the sliding window technique (more precisely, convolving). It moves one pixel to the right each iteration, and once it reaches the end, it starts again on the left side and moves one pixel down. This results in going from 28x28 in the input to 24x24 in the first hidden layer.*

where x is the value in the hidden layer. The function thresholds the layer at 0, resulting in a feature map. The pooling layer comes right after the ReLU and reduces and simplifies the feature map for the next convolutional layer. The most common size of the pooling layer is 2x2, meaning the feature map would go from 24x24 to 12x12. The regularly used pooling method is called max pooling, which outputs the maximum value in the 2x2 grid to a single neuron in the next layer as shown in Figure 2.4. The classifications are usually handled with FC layers at the end of the network, but there have been studies trying to determine the effectiveness of using FC layers over using a SoftMax operation for the classification [13], [20], [37].

2.2.3 Architectures of Convolutional Neural Networks

2.2.3.1 LeNet

The first successful attempt at developing and implementing a CNN architecture started in 1998. It was developed by Yann LeCun, aptly named LeNet [21]. This architecture heavily progressed the field of Deep Learning by laying the groundwork for building architectures. The best known LeNet framework was LeNet-5 developed in 1998 for document recognition, as shown in Figure 2.5. The structure is $INPUT \rightarrow CONV \rightarrow POOL \rightarrow CONV \rightarrow POOL \rightarrow FC \rightarrow FC \rightarrow OUT$. The network was used for zip code and digit recognition in documents, so the output vector is of length 10 containing the probabilities of the input being one of the 10 digits [21].

2.2.3.2 AlexNet

Based off of the LeNet framework and shown in Figure 2.6, Alex Krizhevsky et al. developed

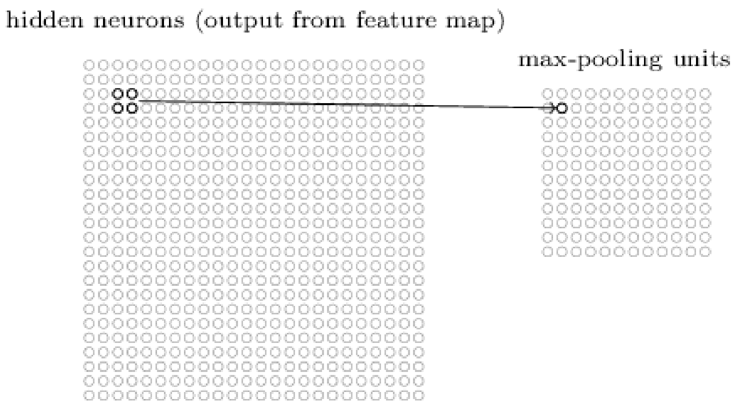


Figure 2.4 Example of a 2x2 max pooling layer being applied to a hidden neuron layer, *from [54].*

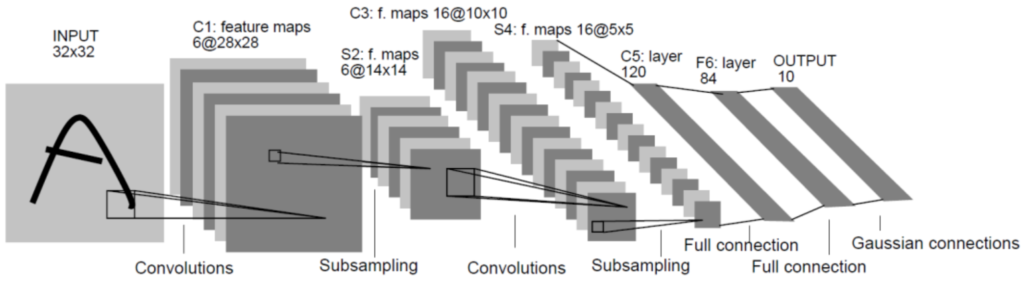


Figure 2.5 Structure of LeNet. *The architecture for LeNet-5 is shown here, from [21]. Each plane is a feature map created from the convolutional layers and the subsampling layers of the network.*

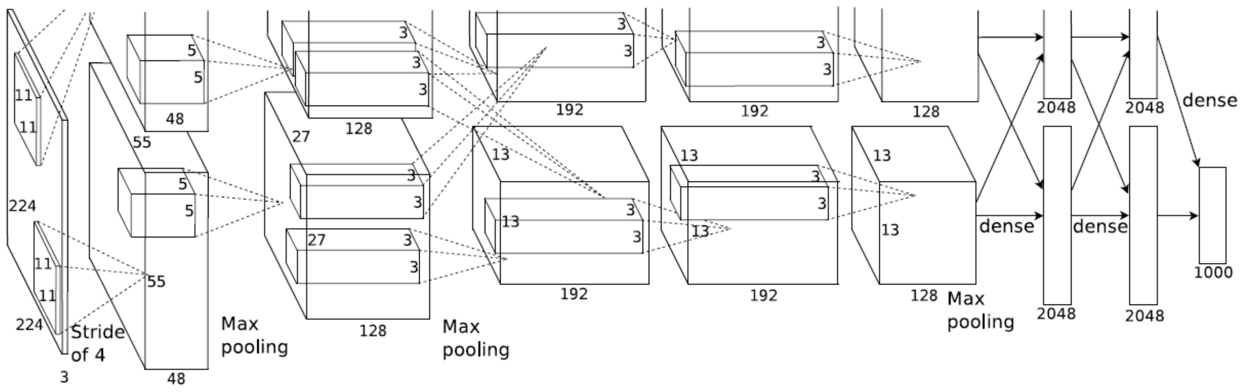


Figure 2.6 Structure of AlexNet. *The architecture for AlexNet is shown here, from [23], [24]. This figure is specifically showing how each of the two GPUs are being used for the network. The top half is one GPU, and the bottom half is the other. The lines between the two halves are when the GPUs communicate, only being on specific layers.*

AlexNet in 2012, which was a much deeper and wider version of LeNet-5 [23], [24]. AlexNet used two GPUs to significantly reduce training time. The network took first place in the very difficult ImageNet Large Scale Visual Recognition Competition (ILSVRC) by a massive margin of 10% error. AlexNet scored a 16% top 5 error while the runner up scored a 26% [26]. This resulted in a revolution in the Computer Vision world by proving how effective CNNs can be for image recognition.

2.2.3.3 ZFNet

With a slight modification to AlexNet, ZFNet was developed by Matthew Zeiler and Rob Fergus, which took first place in ILSVRC 2013 [9]. ZFNet expanded the size of the middle convolutional layers and made the kernel and the stride of the first conv layer smaller to gather more information compared to AlexNet.

2.2.3.4 GoogLeNet

In 2013, Christian Szegedy et al. from Google slightly modified AlexNet to perform better on object detection rather than just classification [11]. Then, they decided to build their own network in 2014, called GoogLeNet, to reduce the computational burden of typical CNNs [20], [37]. Their key goal was to get it to run efficiently on the Google servers at large scale while still achieving state-of-the-art performance. GoogLeNet won first place in ILSVRC 2014. The two main contributions to their architecture were the *Inception module*, shown in Figure 2.7, and using average pooling and a SoftMax classifier instead of the fully connected layers for classification, effectively reducing the operations significantly. There are currently four versions of GoogLeNet, each with slight modifications from the last.

2.2.3.5 VGGNet

Two people in the Visual Geometry Group at the University of Oxford, Karen Simonyan and Andrew Zisserman, created and implemented the CNN architecture that was runner up to GoogLeNet in ILSVRC 2014 called VGGNet, shown in Figure 2.8 [49]. They were the first to use stacked 3x3 convolutional layers throughout the entire network instead of the larger kernels used in previous architectures, which led to being able to simulate a larger receptive field. This meant more complex features could be extracted, and it also proved performance is reliant on the depth of the network. The only downside is the vast number of parameters that comes with using this many small kernels in sequence, ultimately increasing the computational cost for training this network. They made the pretrained network available on multiple different deep learning frameworks to combat this issue.

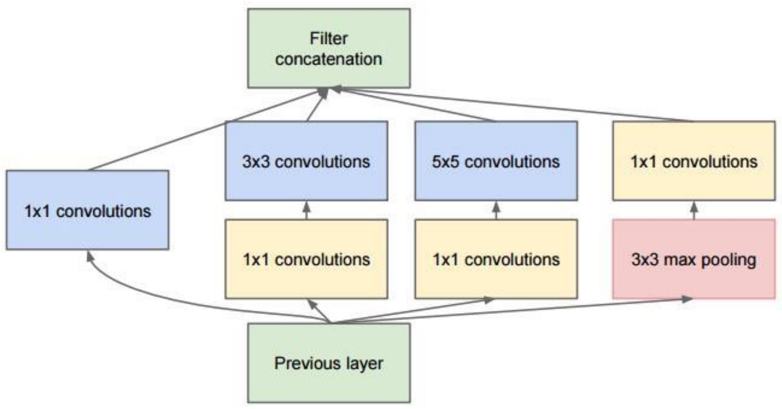
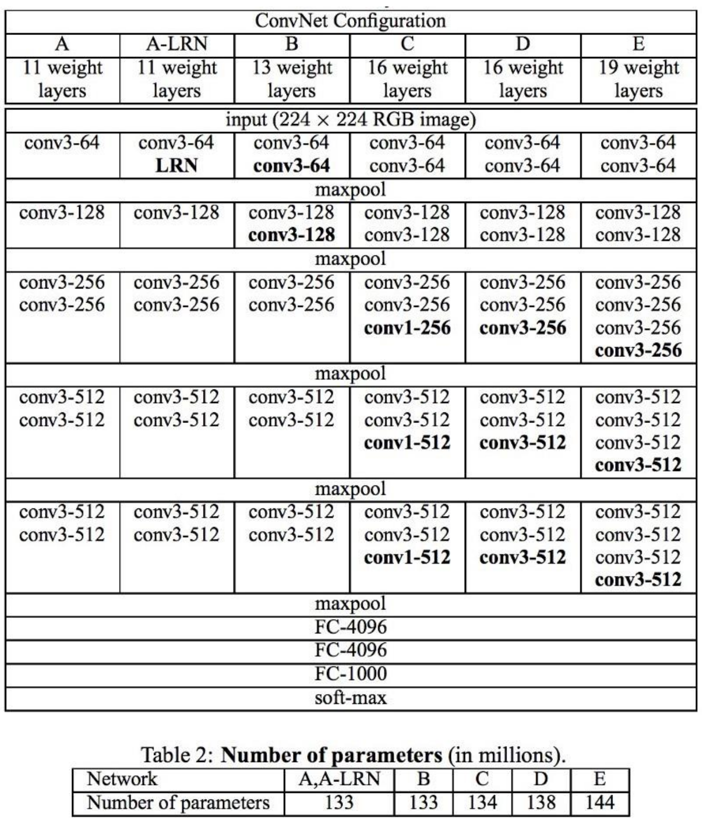


Figure 2.7 Structure of the *Inception module* in GoogLeNet. *The 1x1 convolutions blocks significantly reduce the number of parameters in the network (4m compared to 60m in AlexNet), from [20]. These are used to condense the number of operations having to be applied by each parallel block. Having parallel operations and using 1x1 convolutional blocks help in achieving state-of-the-art performance with a lower computational cost compared to other networks.*



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figure 2.8 Structure of VGGNet. *This shows the 6 different configurations of the network with D performing the best at 16 layers, from [49]. The convolutional layer parameters are denoted as "conv (receptive field size) – (number of channels)". The ReLU activation functions are left out for brevity.*

2.2.3.6 ResNet

The winner of ILSVRC 2015 was ResNet, which was developed by He et al [13]. ResNet achieved an astounding 3.6% error rate, while human error rate is usually between 5% and 10%. Their innovations are skip connections (shown in Figure 2.9), dropping the FC layers off of the end and using a pooling layer then a SoftMax as the final classifier instead, and using substantial amounts of batch normalization. Batch normalization is the technique in which the inputs are normalized before feeding them into the network. In the case of ResNet, the network performs batch normalization in between each convolutional layer and activation function. ResNet was also the first time a network of more than 100 layers was ever trained [13].

2.2.3.7 Summary

There are many different architectures of CNNs that are prominent within the field of deep learning. Most of the aforementioned CNNs have each won awards based on their performance and are the main CNNs used within deep learning models today. We have not performed a fully exhaustive review over all types but wanted to highlight the frequently used ones.

In this research project, we have decided to use the VGGNet as the base CNN within our model. We chose this network based on the accuracy performance of this network and the compatibility with the model we chose – Faster RCNN – as well as based on previous knowledge of using the network. We chose to use Faster RCNN as our object detection model because it was the latest model at the time that showed the highest object detection performance.

2.3 Object Detection using Deep Learning

For the last few years, deep learning architectures have vastly outperformed other methods, such as handcrafted features like Scale-Invariant Feature Transform (SIFT) [15], on various tasks within computer vision, with the most popular being object detection and recognition. The largest

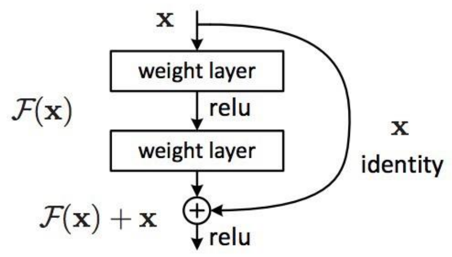


Figure 2.9 Structure of a skip connection used in ResNet. *The skip connection feeds the output of two convolutional layers along with the input into the next layer effectively becoming a small classifier, from [13].*

competition for computer vision tasks is ILSVRC, talked about in Section 2.2. ILSVRC evaluates algorithms for object detection and classification at a very large scale using a subset of the ImageNet dataset, which consists of more than 1.2 million images with 1,000 categories [25]. The competition started in 2010, and since 2012, the winners have been deep learning architectures. This is due to the fact CNNs are able to directly learn object features from large scale datasets, which ImageNet provides. Outside of the competition, there has been progress on creating deep learning models that are used specifically for objection detection and recognition. The models consist of the typical CNN architecture that is pretrained on ImageNet as the core along with another algorithm used either before or after or, in some cases, both before and after to help fine-tune the localization and classification of the object. Next, we will discuss the most prominent models for object detection as well as each one's innovations in the order in which they were developed.

The first ground-breaking model for object detection was developed by Girshick et al. in 2014 called Region-based Convolutional Neural Network (RCNN) [36], [38]. As an alternative to exhaustive search, RCNN uses selective search to propose object containing regions in the image [40]. Selective search initializes around 2,000 small regions in an input image and combines them based on hierarchical measures such as color spaces and similarity metrics. Once all of the regions have been proposed, a CNN is run on top of each one, which equates to running about 2,000 CNNs. The output from all of them are fed into Support Vector Machines (SVMs) to classify the objects and a linear regressor to tighten the bounding boxes. There is one SVM used per output class. The best version of this model achieved 62.4% mAP on the PASCAL VOC2012 test dataset, which was state-of-the-art performance at the time.

In the following year, Girshick was able to identify and address the two glaring problems in RCNN, having to run thousands of CNNs and having to run an SVM for each class. To solve the first problem, he created the technique called Region of Interest Pooling (RoIPool). RoIPool consists of streamlining three distinct functions: applying one CNN to the full input image, applying selective search to the outputted feature maps, and reducing the feature map size using a maxing pooling layer, which output valid RoIs for the input. The second issue was much simpler to address; he substituted the SVM classifiers for a single SoftMax layer for classification. Both of these modifications combined increased the speed considerably over RCNN, and Girshick appropriately named it Fast RCNN [17].

The last bottleneck to address in the RCNN family of models was selective search. In 2016, just one year later, one of the teams at Microsoft Research decided to address this issue by creating the Region Proposal Network (RPN), which uses anchor boxes as well as a sliding window on the feature maps of the CNN to generate an objectness score and the coordinates for each bounding box. The output from the RPN along with the feature maps from the CNN are fed into a Fast RCNN model to perform the classification and adjust the bounding boxes. The team decided to call this model Faster RCNN [18], which is shown in Figure 2.10. It reached much better speeds than Fast RCNN (about 10 times faster) while still maintaining state-of-the-art accuracy. The best version of this model has achieved 78.8% mAP on the PASCAL VOC 2007 test dataset as well as

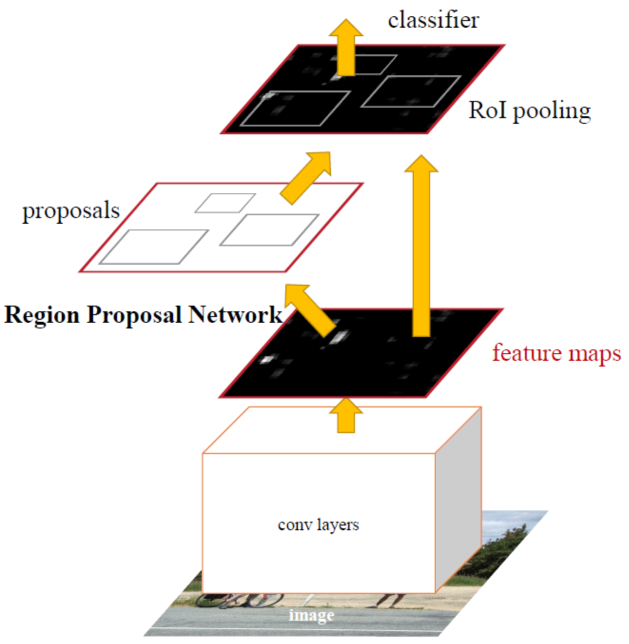


Figure 2.10 Structure of the Faster RCNN model. *This model is a single, unified network for object detection. The RPN module serves as the 'attention' of the model, from [18].*

75.9% mAP on the PASCAL VOC 2012 test dataset. This model concluded the lineage of RCNN models used solely for bounding box detection in images.

Later that year, another model was developed with the sole purpose of being able to detect objects in real time named You Only Look Once (YOLO) [52]. This model predicts bounding boxes and class probabilities with a single CNN in a single evaluation. The model takes an input image and divides it into an S by S grid (default: S = 7). For each cell in the grid, multiple bounding boxes are predicted, each with a confidence score, and non-maximum suppression is applied at the end to merge the overlapping bounding boxes that contain the same class. This results in a model which can perform object detection at 45 frames per second (using a titan X GPU) while maintaining a 63.4% mAP on the PASCAL VOC 2007 test dataset.

At the end of 2016 (same year Faster RCNN and YOLO were released), YOLOv2 was developed by Redmon et al. with the focus of improving the accuracy of YOLO while still maintaining the speed [50]. The new additions in this model include using a slightly larger input image to be able to detect marginally smaller objects, using batch normalization instead of dropout to prevent overfitting, removing the final FC layer to reduce the number of parameters in the model, and adapting the anchor boxes from Faster RCNN. They also decided to use a ResNet like structure to be able to stack both high resolution and low resolution features. Combining all of these modifications, they were able to achieve a 78.6% mAP at 40 frames per second (again on a titan X GPU) on the PASCAL VOC 2007 test dataset when they used both the VOC 2007 and 2012 training and validation datasets to train their model.

A few months later, in March 2017, a research team in Facebook AI developed a model to extend the capabilities of Faster RCNN, called Mask RCNN, to be able to perform image segmentation, which is locating each pixel of each object within an image [30]. The key innovation of this model was called RoIAlign, which computes the subpixel values of the location of pixels within the feature maps and uses those calculations for the region proposals. This subpixel calculation allows for more precise bounding box detections, which leads to being able to associate each pixel with an object. Mask RCNN outperformed the current state-of-the-art model in multiple different challenges.

In 2018, YOLOv3 was created by Redmon et al. trying to address the main complaint about YOLOv2 not detecting small objects [51]. The new model trades speed for accuracy, which is accomplished by adding more layers and performing the object detection at three separate places within the network. In total, there are 106 layers being used with the last 53 of those being used specifically for object detection. The three detections are made at three different sizes in the network, which, in turn, help with detecting small, medium, and large objects. The model also uses 3 anchor boxes per detection (9 in all), which leads to 10-12x more bounding boxes than YOLOv2. This is one of the reasons this version is slower. YOLOv3 runs at 30 frames per second.

2.4 Sensitivity Analysis

In this section, we discuss a general overview of the degradation problem, previously attempted solutions, how a sensitivity analysis can be used to mitigate this problem, how one is performed, and then, we review the current state of the art in sensitivity analysis applied to object detection deep learning models. We will also discuss what will be performed in this thesis, and why this is an extension to the previous works.

In the context of computer vision, performing a sensitivity analysis is applying degradation techniques to images before testing a deep learning model and analyzing the results to determine how robust the model is. Currently, deep CNNs are trained on pristine images, implying there is an assumption the test images will also be pristine with no artifacts. This means the model will not perform well on images taken by a mobile phone or a non-high quality camera, which would make it useless for everyday images. One sub-optimal solution to this problem is using denoising methods as a preprocessing step in the model, which has proven to not work effectively unless a specific form of noise has been applied to the images beforehand [16], [48]. In a general sense, denoising methods typically consist of smoothing out the image while trying to maintain the edges and the fine details of objects. The main problem with these methods is they do not generalize to other types of noise well. If a test set contains a mixture of naturally noisy images, pristine images, and images with a specific form of noise applied to them, a denoising method used as a pre-processing step would most likely prove useless while extending the time it takes for the model to test. This is because the artifact-free images would be fed through the denoising method even though it is not needed, providing poor results. The naturally noisy images would also provide poor results, since most denoising methods cannot effectively reduce natural noise [16], [48]. Sensitivity analysis, as it is being used in this project, can be a viable method for testing the robustness of the deep learning model to determine how effective the model is when given low quality images.

In previous works, a relatively small number of degradation techniques have been applied to images before model inference has been performed to test the model robustness. In [48], the authors applied blur, noise, contrast scaling, and JPEG compression to the dataset they used. These techniques were used in such a way that humans would still be able to identify the objects within the images. Their results showed how each of these methods significantly decreased the performance of their model and how this issue cannot be addressed properly by existing convolutional layers. Blur, noise, and JPEG compression – a smaller subset of the aforementioned techniques – were used in [22] showing similar results on the degraded imagery as [48]. The unique approach they applied to the models is image quality assessment based label smoothing. This proved to significantly improve the results of the model, although this only applies to image classification, not object detection. In addition, [16] decided to take it one step further and apply various levels of three different types of noise as well as multiple levels of JPEG compression to their images. Their results were very similar to [48], so they created a version of denoising to use on the images as a preprocessing step before testing the model. Although, the results were better than if no denoising method was used, their results proceeded to not have a significant enough impact on the performance to warrant using an extra step in the model. Yim et al. concluded the method they used lost too many details in the smoothing process since deep learning models rely on more than just the object edges to perform well.

In this research, we are going to use nine degradation functions to perform our sensitivity analysis, which is more than the previous papers applied to their image sets. We wanted to use the same methods the previous papers used as well as adding more functions to increase the variation in the overall results. The functions we chose are changing the amounts of brightness, saturation, contrast, Gaussian blur, image size, sharpness, Gaussian noise, speckle noise, and salt and pepper noise. The reason we chose this many functions is to produce more results than the previous works and allow us to more accurately determine which functions the model is more robust towards and which have greater impacts. Using this many functions within a single sensitivity analysis has not been performed by any other researchers at this time, which makes this thesis unique in that aspect.

3 Training and Sensitivity Analysis

The main goal of this project is to apply the Faster RCNN model to a custom dataset and to perform a sensitivity analysis on the model. The objectives of executing these two tasks are (1) to observe the performance of Faster RCNN given a new object class to detect and (2) to assess the robustness of the model on artificially degraded imagery. The following sections are divided into four parts: applying the model to the binary class dataset, applying the model to the multi-class dataset, an example of applying the model to the binary class dataset combined with multi-class, and the sensitivity analysis on the VOC2007 dataset.

3.1 Training Faster RCNN using Aphylla

In this research, we chose to use the Faster RCNN model to perform object detection on our binary class custom dataset. This is because Faster RCNN is more accurate compared to the competition [43], and our focus was on accuracy and not speed. The implementation of the model we used in this project uses VGG16 as the base network [35]. This network was provided in the torchvision package within PyTorch as a network pre-trained on ImageNet. PyTorch is a deep learning framework that is built using the Python coding language [60]. Torchvision includes various models and popular datasets that can be used freely within PyTorch to train and test different networks. Both the architecture of VGG16 and the ImageNet dataset were described in chapter 2.

The implementation of Faster RCNN we used was ported to PyTorch from another framework. In that process, [3] decided to make a few modifications for the ease of translation. The major modifications made were three-fold: (1) instead of using RoIPool, they used *crop_and_resize*, a new module they created; (2) they decided not to aggregate the gradients; and (3) they decided to keep the small region proposals. Following the work of [43], the *crop_and_resize* module is used in the place of RoIPool in the model. This module crops and resizes the resulting feature map that would normally be fed into RoIPool down to the proper size and performs max pooling to resize the feature map, which is then fed into the fully connected layer for classification. This change in modules appears to have a slight advantage over the original. The next modification made is not aggregating the gradients, which requires extra operations when ported into PyTorch. These extra operations noticeably slow down the model. The last major change is the decision to keep the small region proposals, which were originally discarded if they were under 16 pixels in height or width. This allows the new model to be able to detect marginally smaller objects. Overall, these modifications to the original model only slightly effect the performance. Refer to [3] to view the rest of the modifications and the results.

3.1.1 Modifications to Run Faster RCNN

Many modifications needed to be made on the model for our dataset to work. The first pre-processing step we needed to make was to randomly split our custom dataset into training, validation, and testing sets. We decided to use 50%/25%/25% for our ratios since they closely relate to the split of the original VOC2007 dataset, which is 25%/25%/50%. The only difference is we decided to make the training set larger than the validation and testing sets to ensure our model would have enough data to train on. Another step that needed to be made was converting the annotations for the images to the proper format the model requires, which is based on the original VOC2007 dataset. When converting the annotations, we discovered VOC uses 1-based indexing while Aphylla uses 0-based indexing for the starting points of the bounding boxes. This means the bounding boxes in our dataset were not being correctly interpreted by the model. Once we fixed this error, we started getting correct results. The other modifications have been adapted from [62], [63]. These two solutions have been significantly helpful in the process of getting Faster RCNN to work on our custom dataset.

3.1.2 Model Training with Aphylla

Using the custom dataset, we trained and tested the Faster RCNN model using the default parameters found in [18], [3]. The key parameters from those papers are the batch size, the loss function, the optimizer, and the epochs. The batch size is one image instead of a typical batch size greater than one, e.g. [20], [24], [52] etc. This is to account for the model being able to train on varying sizes of images without having to pad the edges of the images to make them the same size. The next significant parameter is the cross entropy loss (CEL) function. This function calculates the loss, which is the performance of classifying the objects. The equation for CEL is as follows:

$$CEL = -\sum_{c=1}^{M} y_{o,c} \log\left(p_{o,c}\right) \tag{3.1}$$

where M is the number of classes; log is the natural log; y is the binary indicator based on if c, the class label, is correct based on o, the observed class; and p is the predicted probability o is of class c [61]. Next, the model tries to minimize the CEL using stochastic gradient descent (SGD), which is the optimizer. The SGD algorithm in this model uses an initial learning rate (LR) of 0.001 with a momentum of 0.9. The LR is decayed by 0.1 to 0.0001 after 30,000 epochs. SGD is the main contributor to the model being able to learn to correctly detect objects. The last main parameter we want to mention is the number of total epochs we train the model for, which is 70,000. This is the default number of epochs used to fine-tune Faster RCNN on the original dataset [3], and we decided to use this with our custom dataset since it produced notable results.

3.2 Training Faster RCNN using VOC2007

The next task we needed to perform before we could start on the sensitivity analysis is training Faster RCNN on the VOC2007 dataset, which is the multi-class dataset we decided to use. We wanted to produce our own trained model on this dataset rather than using the trained model provided by [35]. This proved to be a much simpler task than training on the binary class dataset since the model was already adjusted to train properly on the VOC2007 dataset beforehand. We used the default parameters as stated in [3] to train the model. We also used the default number of epochs stated in [3], which is 70,000. This worked without any issues and produced a fully trained Faster RCNN model.

3.3 Dataset used for Sensitivity Analysis

The sensitivity analysis performed in this project determines the robustness of the Faster RCNN model on degraded images. Deep learning models, in general, are trained on pristine images to detect objects, meaning the model will perform poorly when tested on naturally degraded images, i.e. from a smartphone or a non-high quality camera, [16], [48]. The techniques we chose to use will simulate naturally degraded images and will allow us to determine how degraded an image can be before the performance of the model significantly decreases. In the following sections, we will describe how the number of images in the image set was chosen, present the equations and ranges for each degradation technique with examples of each, and explain how we tested the model on these degraded image sets.

A smaller test set size needed to be chosen for the VOC2007 dataset to reduce the computational time in both applying each degradation function to the test set and testing the model. To accomplish this task, we started by creating test sets beginning at 1,024 images and decreasing by a factor of two to 64. We chose to use the Gaussian blur function as the technique to apply to each set since our preliminary results showed the output of the model when using this function had a more predictable output curve compared to the other techniques. In our attempt of finding the best test set size, we graphed the mean average precision (mAP) scores for each size and compared them to determine how small the set could be while still maintaining within 5% error rate of the mAP scores for the entire set – 67.6% mAP using our 512 testing set versus the baseline score of 71.1% mAP using the entire 4,952 image set. This turned out to be 512 images, which also allowed us to reduce the computational time significantly as compared to the original test set size of 4,952 images.

Once we chose 512 images as the optimal set size, the images contained in the initial random sample of 512 needed to be checked for any bias. A bias in the smaller dataset would result in the object detections not closely matching other randomized 512 image datasets. To check for this, we randomly sampled 1,536 images from the overall test set, divided them into three 512 image sets, and tested the model on each set with visualized results, as shown in Figure 3.1. The comparison showed that the three sets we created were all within one standard deviation of each other, which concludes there is no bias.

3.4 Degradation Techniques used for Sensitivity Analysis

In this project, we used degradation techniques to attempt to cover the majority of possible artifacts found naturally in noisy images. The techniques include adjusting nine different functions: brightness, saturation, contrast, gaussian blur and noise, image resizing, sharpness, speckle noise, and salt and pepper noise. These functions are applied separately to each image set, and then the

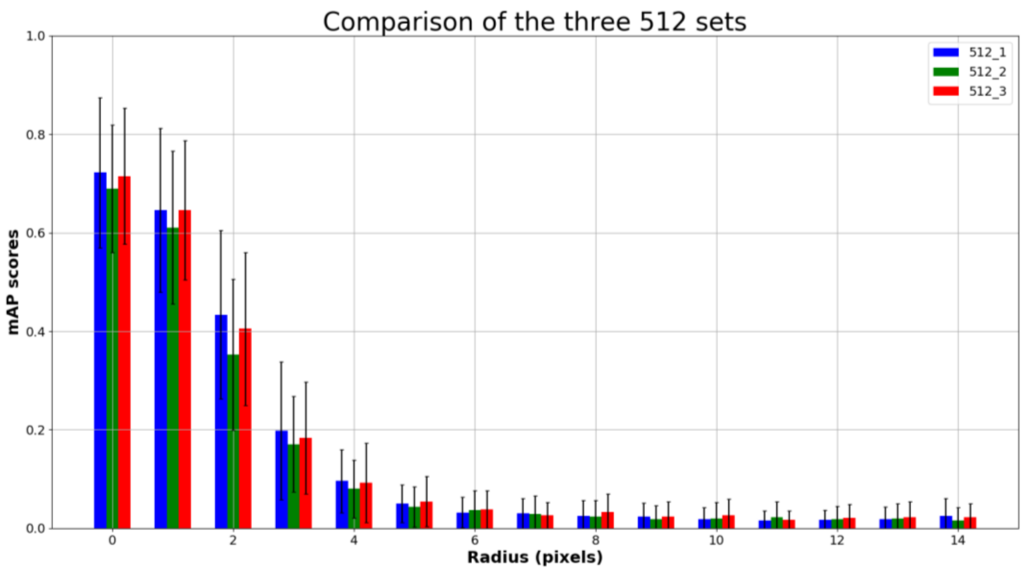


Figure 3.1 Comparison between the three 512 image datasets to determine bias. *This graph is of the results of creating three 512 image test sets, applying Gaussian blur to each image, and then getting the results of the model. This graph shows the mAP scores vs. the Gaussian blur radius in pixels. The error bars on each bar are one standard deviation above and below the mAP score.*

model is tested on each individually.

To perform an effective sensitivity analysis, one of the first things we needed to consider was how to choose the ranges for each function. For consistency purposes, we created a set of criteria for choosing the ranges, which stated the ranges needed (1) to include the extrema for each technique and (2) to be significantly visually different from image to image. The extrema for each technique are the minimum value for each range and the maximum value. In some cases, we only were able to get as close as possible to the extrema since the actual extrema points were infeasible. This will be explained in the next part of this section.

The second criterion we needed to meet was to make sure the ranges contained an adequate amount of visual differences in each iteration. We did not want to space out each step too close or too far from one another, in other words, trying to cover the best possible range for each technique. For us to achieve this, the ranges needed to be a mixture of linear spacing and log spacing for a couple of reasons. First, we had to make sure to include the original image within the range and to include both the extrema. The second reason is we had to make sure each step in the ranges had enough visual differences to warrant including them. We did not want wasted space in each range. We also made sure these rules did not produce duplicates in the ranges, since duplicates would add to the computational time in both creating the image and testing the model. Along with these two standards, we ultimately decided to make each range contain 12 different steps to ensure proper coverage of each technique. All of this allowed us to obtain meaningful results when testing the model.

3.4.1 Brightness

The first technique we apply to the image set is adjusting the brightness. This allows us to generate images that are ranging from pitch black to white-washed, which attempt to replicate a type of non-pristine image. We adjust the levels of brightness by using the following transformations:

$$\begin{aligned} g_r(x,y) &= f_r(x,y) * ratio \\ g_g(x,y) &= f_g(x,y) * ratio \\ g_b(x,y) &= f_b(x,y) * ratio \end{aligned} \tag{3.2}$$

$$ratio = [0\%, 3000\%]$$

where $g(x,y)$ is the output RGB values, $f(x,y)$ is the input RGB values, and $ratio$ is the desired brightness over the original brightness of the image [55], [56]. The $ratio$ values used for the first portion of the range start at 0% and go to 100% using the log space, which allowed us to obtain more images with low brightness and fewer with close to original brightness. The second set of $ratio$ values start at 100% and go to 3000% using the log space again. Breaking the values into two sets allowed us to make sure the original image (at 100%) would be included and to have more images at lower brightness levels than higher, which would meet the second criterion. An example of this range of brightness is shown in Figure 3.2. The range of $ratio$ values is 0%, 10%, 15%,

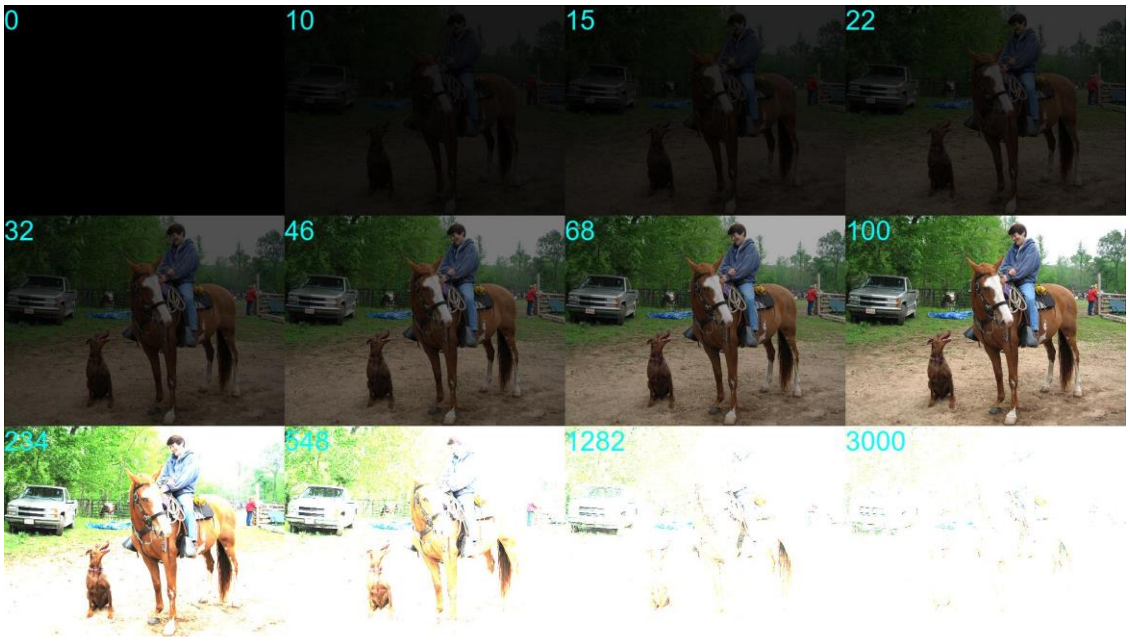


Figure 3.2 Example image displaying the various levels of brightness, indicated in the top left in cyan as a percentage. *Brightness levels ranging from 0% (all black) to 3000% (white-washed), including 100% (untransformed).*

22%, 32%, 46%, 68%, 100%, 234%, 546%, 1282%, and 3000%.

3.4.2 *Saturation*

The second technique we utilize is adjusting the saturation levels in the image. This task is accomplished by, first, calculating $g_p(x,y)$, pixel luminance:

$$g_p(x,y) = 0.299 * f_r(x,y) + 0.587 * f_g(x,y) + 0.114 * f_b(x,y) \tag{3.3}$$

where $f_r(x,y)$ is the red value, $f_g(x,y)$ is the green value, and $f_b(x,y)$ is the blue value of each pixel in the image. The second equation, which is directly controlling the saturation and returns the new RGB values, is as follows:

$$\begin{gathered} g_r(x,y) = f_r(x,y) - \left(\left(f_r(x,y) - g_p(x,y)\right) * ratio\right) \\ g_g(x,y) = f_g(x,y) - \left(\left(f_g(x,y) - g_p(x,y)\right) * ratio\right) \\ g_b(x,y) = f_b(x,y) - \left(\left(f_b(x,y) - g_p(x,y)\right) * ratio\right) \\ ratio = [0\%, 750\%] \end{gathered} \tag{3.4}$$

where $ratio$ is the desired saturation over the original saturation in the image [55], [56]. Saturation is the amount of gray mixed in with the pure colors in each pixel. A highly saturated image would contain more pure colors with less gray in them while an undersaturated image would tend to be grayer overall. As shown in Figure 3.3, the range used for adjusting the saturation attempts to cover the full range from grayscale to heavily saturated. The first set of $ratio$ values start at 0% saturation, which is a grayscale image, and goes to 100% (untransformed). This section of the range uses linear spacing, increasing by 25%. The second part of the range starts at 100% and goes to 750% using the log space. This allowed us to accurately depict the higher saturation values while also maintaining a visual difference in each image. The range of $ratio$ values is 0%, 25%, 50%, 75%, 100%, 133%, 178%, 237%, 316%, 422%, 562%, and 750%.

3.4.3 *Contrast*

The next technique we apply to the images is adjusting the contrast level. Contrast in an image is the difference between the maximum and minimum pixel values. In other words, it determines how distinguishable an object is within the image. Maximum contrast would mean every object in

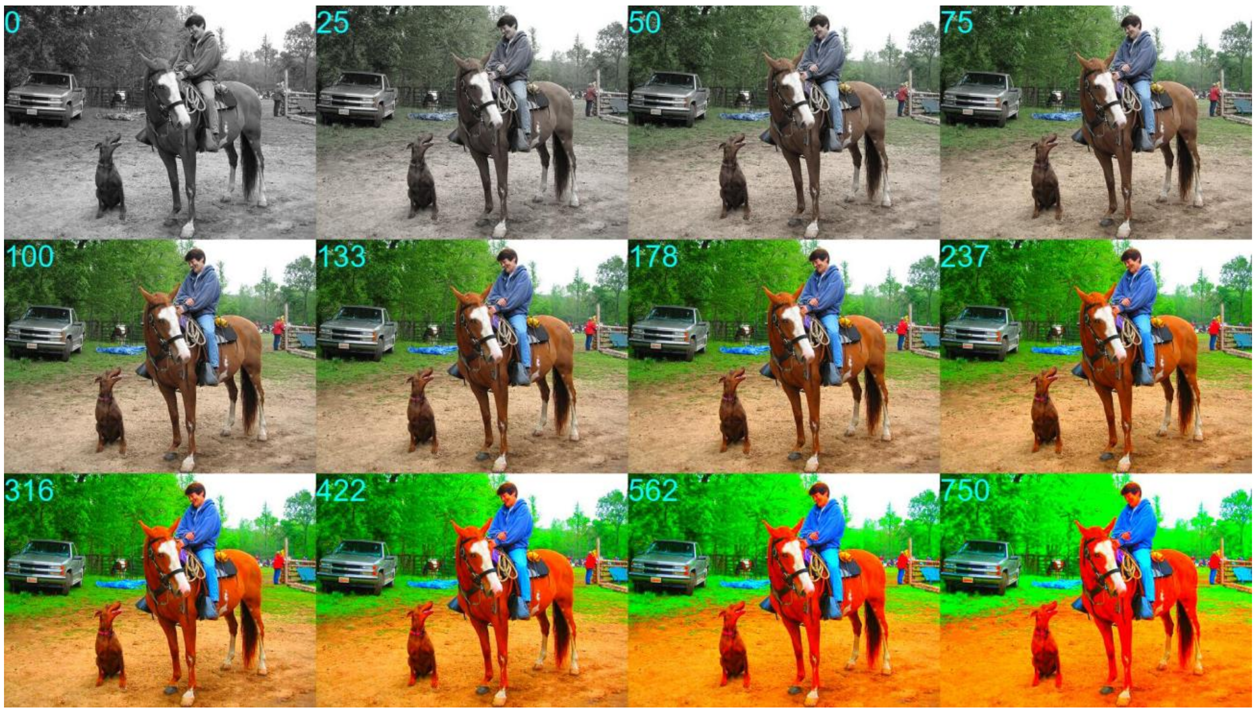


Figure 3.3 Example image displaying the various levels of saturation, indicated in the top left in cyan as a percentage. *Saturation levels ranging from 0% (grayscale) to 750% (heavily saturated), including 100% (untransformed).*

the image significantly stands out with all the colors overly exaggerated – much deeper than normal. Minimum contrast would mean the image is completely gray. The equations used to adjust the contrast levels are as follows:

$$factor = \frac{259 * (c + 255)}{255 * (259 - c)} \tag{3.5}$$

$$\begin{aligned} g_r(x,y) &= factor * (f_r(x,y) - 124) + 124 \\ g_g(x,y) &= factor * \left(f_g(x,y) - 124\right) + 124 \\ g_b(x,y) &= factor * (f_b(x,y) - 124) + 124 \end{aligned} \tag{3.6}$$

where,

$$c = \begin{cases} ratio * 255; \; if\ ratio \geq 1 \\ ratio * -255; \; if\ ratio < 1 \end{cases} \tag{3.7}$$

$$ratio = [0\%, 2000\%]$$

and $ratio$ is the desired contrast over the original contrast [39]. Equation 3.5 is the contrast correctional factor equation, which allows the contrast to be user controlled with a simple ratio percentage [56]. The value of 259 in the equation is an experimental number the creators of this particular python package found to work nicely within the equation [56]. In the equations calculating the new RGB values, 124 is used as the middle value of the image or pure gray. The middle value is typically 127, since that is the true gray value, but the package we used in Python put its own mark on the equation [56]. The first portion of the values used in adjusting the contrast starts at 0% and goes to 100% using linear spacing. The second set starts at 100% and goes to 2000% using log space. Using this range allowed us to have a greater amount of differences in each image, as shown in Figure 3.4. The $ratio$ values used in this range are 0%, 5%, 21%, 37%, 52%, 68%, 84%, 100%, 211% 447%, 946%, and 2000%.

3.4.4 Gaussian Blur

After adjusting different image intensity factors, we decided to apply a Gaussian blur to the image set. This simulates viewing an image through a semi-transparent screen, which makes the entire image appear out of focus. The equation used for applying Gaussian blur is as follows:

$$g_{GB}(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.8}$$

$$\sigma = [0, 9]$$

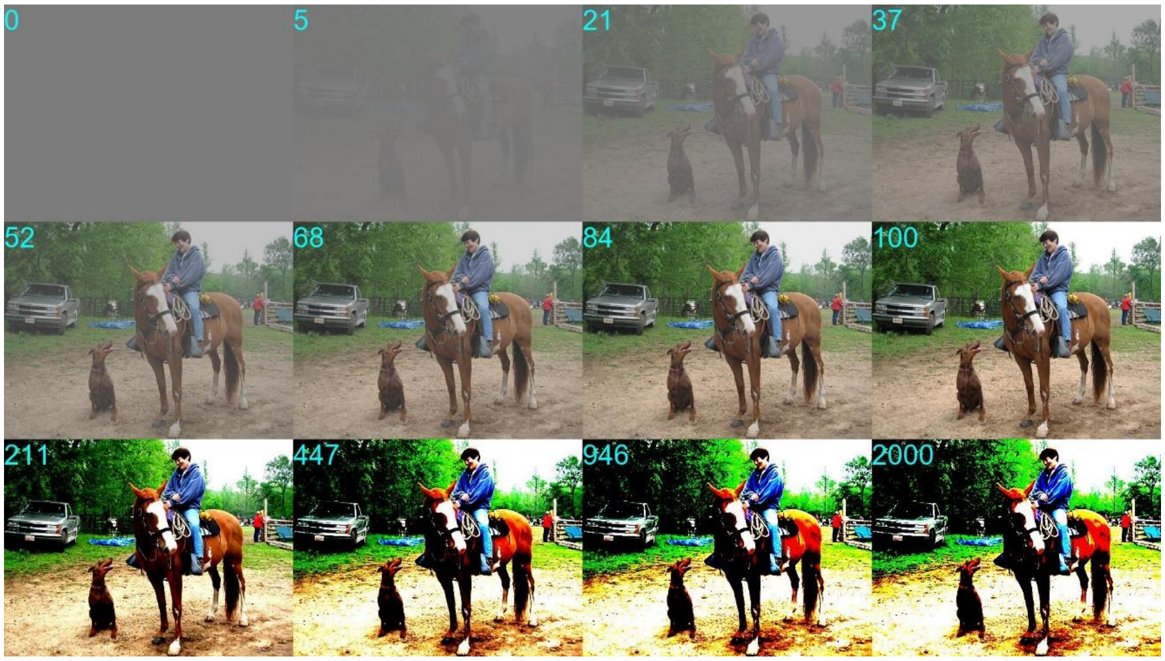


Figure 3.4 Example image displaying the various levels of contrast, indicated in the top left in cyan as a percentage. *Contrast levels ranging from 0% (all gray) to 2000% (heavily contrasted), including 100% (untransformed).*

where x is the distance from the origin on the x-axis, y is the distance from the origin on the y-axis, and σ is the standard deviation (blur radius) [55], [56]. The blur radius is the variable factor in this equation that controls how much blur the resulting image will have. The range used in this technique starts at 0 pixel radius and goes to 2 in increments of 0.5, and then starts again at 3 and goes to 9 in increments of 1, as shown in Figure 3.5. The increments of half of a pixel in the first set of values are considered a sub-pixel, which is still calculatable as a blur radius. This range allowed us to demonstrate the differing amounts of blur that could be present in an image. The blur radius values used are 0, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, and 9.

3.4.5 Image Size

The next technique we used is resizing the image and scaling it back up to the original height and width. This simulates a pixelated or low resolution image. Resizing the image with a small percentage based off the original size would result in the image containing very few pixels. A large percentage would be mostly unnoticeable since the image would only be resized slightly. Instead of using an equation, this technique uses an algorithm to resize the image with the variable being the percentage of the original image size [56]. We tested multiple different resampling filters, but ultimately decided to use the nearest neighbor filter. This resampling filter simulates a lower resolution image better than the others. The range used for the scaling percentage started at 100% and decreased to 4% in log space as this allows for more images on the lower side of the range. An example image with this range applied is shown in Figure 3.6. The range of levels of image resizing is 100%, 75%, 56%, 42%, 31%, 23%, 17%, 13%, 10%, 7%, 5%, and 4%.

$$g_{is}(x, y) = resize(f(x, y) * ratio)\ using\ Nearest\ Neighbor\ resampling$$

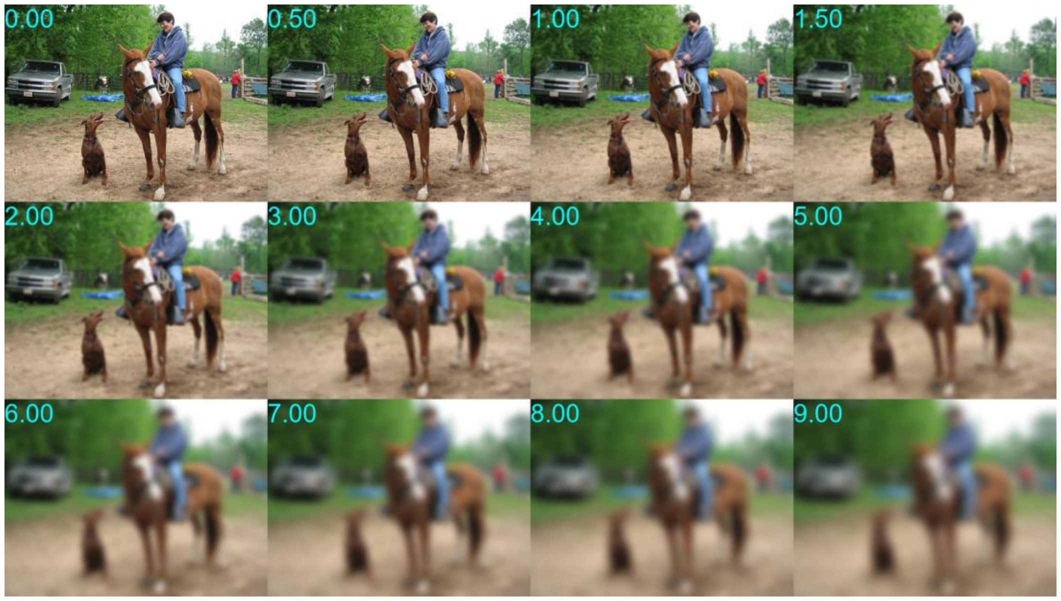


Figure 3.5 Example image displaying the various radii of Gaussian blur, indicated in the top left in cyan. *Gaussian blur with radii ranging from 0 (untransformed) to 9 pixels (heavily blurred).*

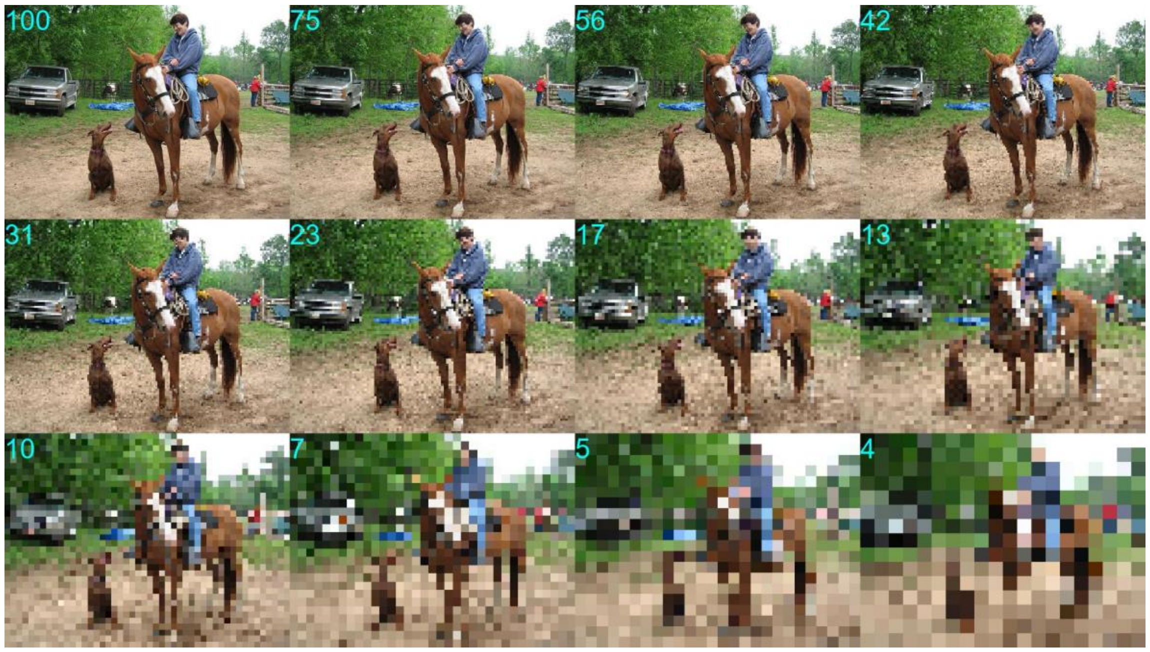


Figure 3.6 Example image displaying the various levels of image resizing, indicated in the top left in cyan as a percentage. *Sizes ranging from 100% (untransformed) to 4% (pixelated). Resizing replicates pixelization by reducing the image by a given percentage of the original image size using the nearest neighbor filter, then upscaling the image to the original size.*

3.4.6 Sharpness

Image sharpness is related to image intensity as well, but it is not as directly related as the first three image characteristics mentioned above. For this technique, we adjust the sharpness within the image, which is the equivalent of enhancing the edges of the objects. At 0% sharpness, the resulting image is the original. At increasingly higher percentages, the output image has artificially enhanced edges around every object in the image. The calculation used for this is as follows:

$$g_s(x,y) = f(x,y) + \big(f(x,y) - g_{GB}(x,y)\big) * amount \tag{3.9}$$

$$amount = [0\%, 374\%]$$

where $g_s(x,y)$ is the sharpened image, $f(x,y)$ is the original image, $g_{GB}(x,y)$ is the Gaussian blurred image with radius = 2, and $amount$ is the percentage of sharpness desired [55], [56]. The values we used for adjusting the ratio of sharpness start at 0% and go to 374% using a linear spacing, as shown in Figure 3.7. We chose these values to attempt to depict visual differences throughout the range. These values were the most difficult for us to attempt to apply the criteria to, because there are not many visual differences at higher percentages of sharpness as it is difficult

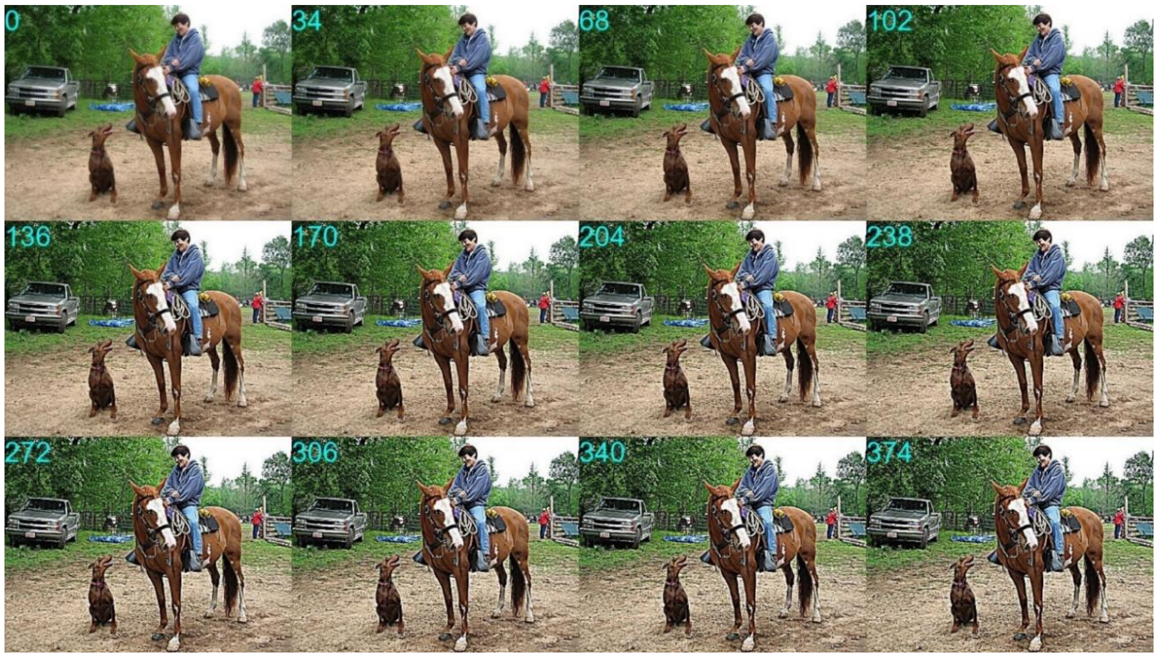


Figure 3.7 Example image displaying the various levels of sharpness, indicated in the top left in cyan as a percentage. *Sharpness levels ranging from 0% (untransformed) to 374% (over-sharpened).*

for a human viewer to differentiate them. The $amount$ values used in this range are 0%, 34%, 68%, 102%, 136%, 170%, 204%, 238%, 272%, 306%, 340%, and 374%.

3.4.7 Gaussian Noise

The last artifact typically found in images that we decided to manipulate is adding different types of noise. We chose three different variations: Gaussian noise, speckle noise, and salt and pepper noise. These three types help simulate noisy images taken by non-high quality or malfunctioning cameras. The equations for applying the first type of noise – Gaussian noise – to the images are as follows:

$$p_G(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{3.10}$$

$$g_{GN}(x,y) = f(x,y) + p_G(z) \tag{3.11}$$

$$\sigma^2 = [0, 2.56]$$

where $p_G(z)$ is the Gaussian noise output, z is the gray level of the current pixel (uses luminance equation stated above in saturation), μ is the mean value (default: 0), σ^2 is the variance, and $f(x, y)$ is the original image [39]. Variance is the variable factor for this technique, while the mean value was left at its default. The resulting noise in the images follows the statistical normal distribution, which is also called the Gaussian distribution [55]. The range used for adding Gaussian noise started at 0 variance, which is the original image, and increased to 2.56 variance using the log space, as shown in Figure 3.8. The log space allowed the range to contain more images with lower amounts of noise. Higher amounts of noise are harder for a human viewer to visually differentiate between, which would have meant using a linear spacing would have broken our criteria since this would have contained a higher quantity of noisier mages. The range used for the $variance$ is 0, 0.01, 0.02, 0.03, 0.05, 0.09, 0.16, 0.28, 0.49, 0.84, 1.47, 2.56.

3.4.8 Speckle Noise

Speckle noise is inherently found in radar images and medical ultrasound images [1], [41]. Given this, we decided to add multiple levels of speckle noise to the image set to simulate the levels found in the aforementioned types of images. The equations used for adding speckle noise are as follows:

$$p_G(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{3.12}$$

$$g_{Sp}(x, y) = f(x, y) + \big(f(x, y) * p_G(z)\big) \tag{3.13}$$

$$\sigma^2 = [0, 15]$$

where $p_G(z)$ is calculated using Equation 3.10, z is the grey level of the current pixel (uses luminance equation stated above in saturation), μ is the mean value (default: 0), σ^2 is the variance, $f(x, y)$ is the original image [39]. Like in the Gaussian noise equation, variance is the variable we changed to control how much noise is present in the image, which ranges from 0 to 15 as shown in Figure 3.9. The range of $variance$ values is 0, 0.03, 0.06, 0.1, 0.19, 0.36, 0.67, 1.25, 2.32, 4.33, 8.06, and 15.

3.4.9 Salt and Pepper Noise

The last technique is adjusting the amount of salt and pepper (S&P) noise in the images, also known as impulse noise. This type is commonly found in images produced by a faulty memory storage device or a faulty camera sensor [14]. We use this technique to simulate images taken while using the faulty equipment mentioned above. There is no simple equation for this type of noise. It is implemented with an algorithm, which can be found in [39]. S&P noise is randomly

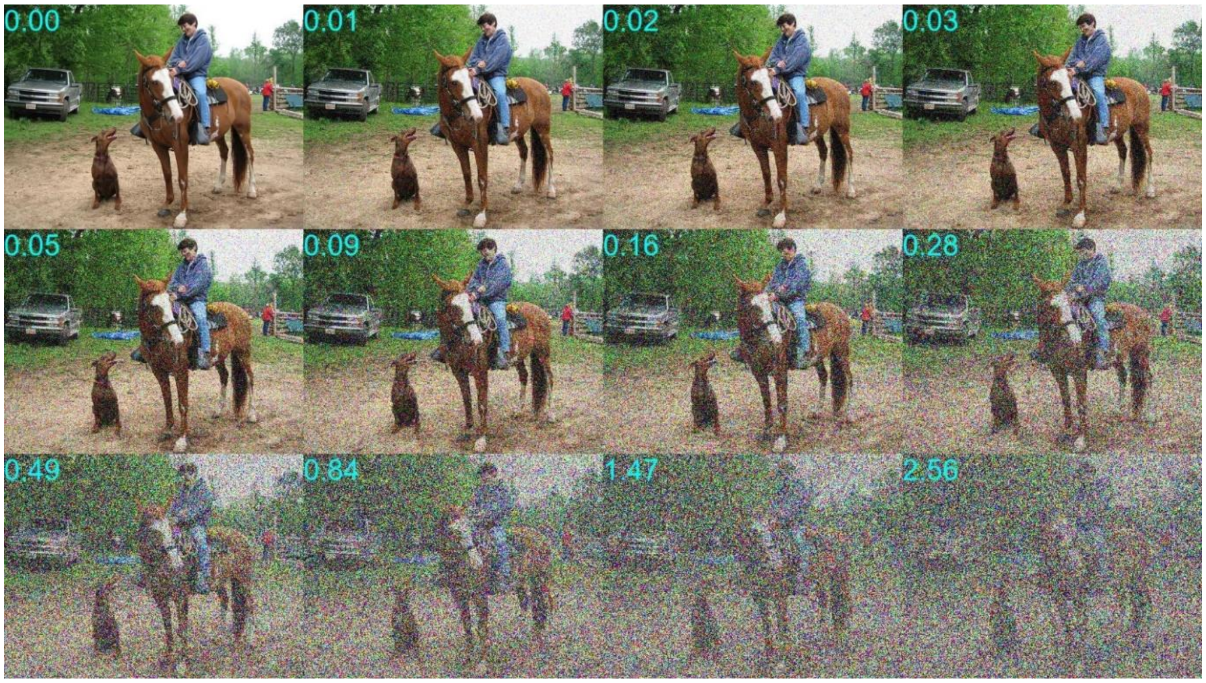


Figure 3.8 Example image displaying the various levels of Gaussian noise, indicated in the top left in cyan as the variance. *Gaussian noise with variance ranging from 0 (untransformed) to 2.56 (very noisy).*

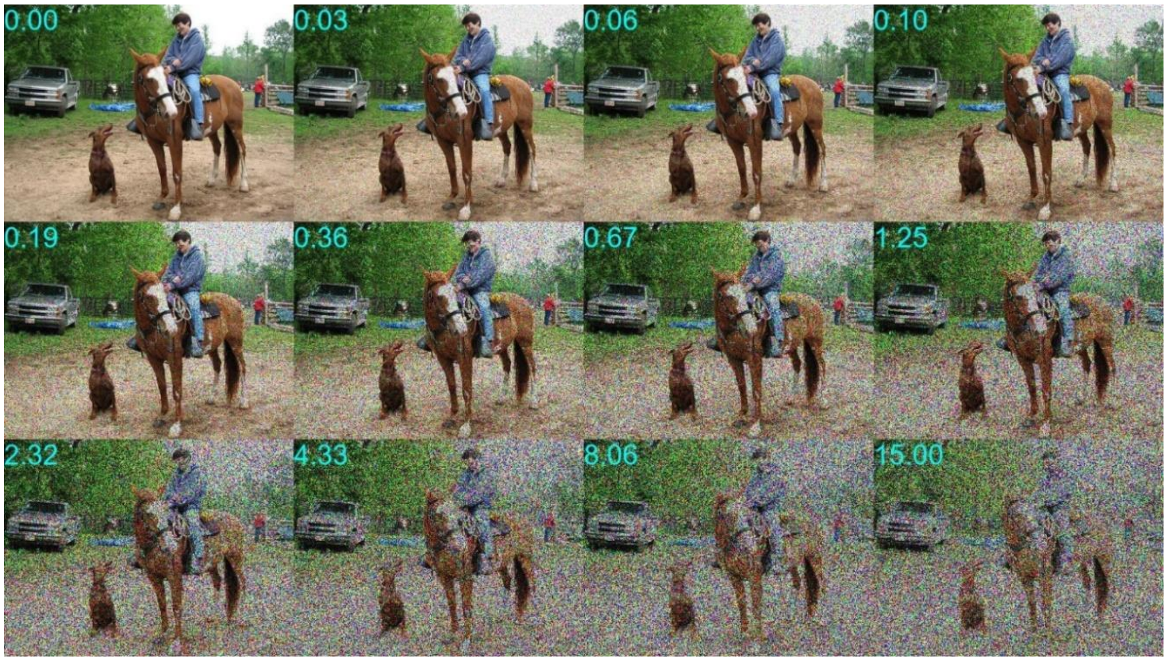


Figure 3.9 Example image displaying the various levels of speckle noise, indicated in the top left in cyan as the variance. *Speckle noise with variance ranging from 0 (untransformed) to 15 (very noisy).*

distributed within the image and can only be the minimum (pepper) or the maximum (salt) values in the typical image range of [0, 255]. We manipulated the amounts of S&P noise within the images using a random, uniform distribution of half salted pixels and half peppered pixels based on a given percentage using [39]. The range for this starts at 0% noisy pixels and increases to 50% using linear spacing as shown in Figure 3.10. The percentages used in this range are 0%, 5%, 9%, 14%, 18%, 23%, 27%, 32%, 36%, 41%, 45%, and 50%.

$$g_{snp}(x,y) = \begin{cases} 0, for\ random(0.5 * ratio * f(x,y)) \\ 255, for\ random(0.5 * ratio * f(x,y)) \end{cases}$$

$$ratio = [0, 0.5]$$

3.5 Model Testing – Degraded Image Sets

We applied the model onto each resulting image set to test the robustness using the default model parameters provided in [18]. In total, there are 55,296 images split into 108 different image sets. These were created from the nine different functions with 12 values in each range. We decided to combine all of the images into one image folder, so we would be able to test the model 108 different times in series. Using this method for testing allowed all the results to be stored in separate log files, which enabled us to graph and compare each iteration of each technique separately. If we

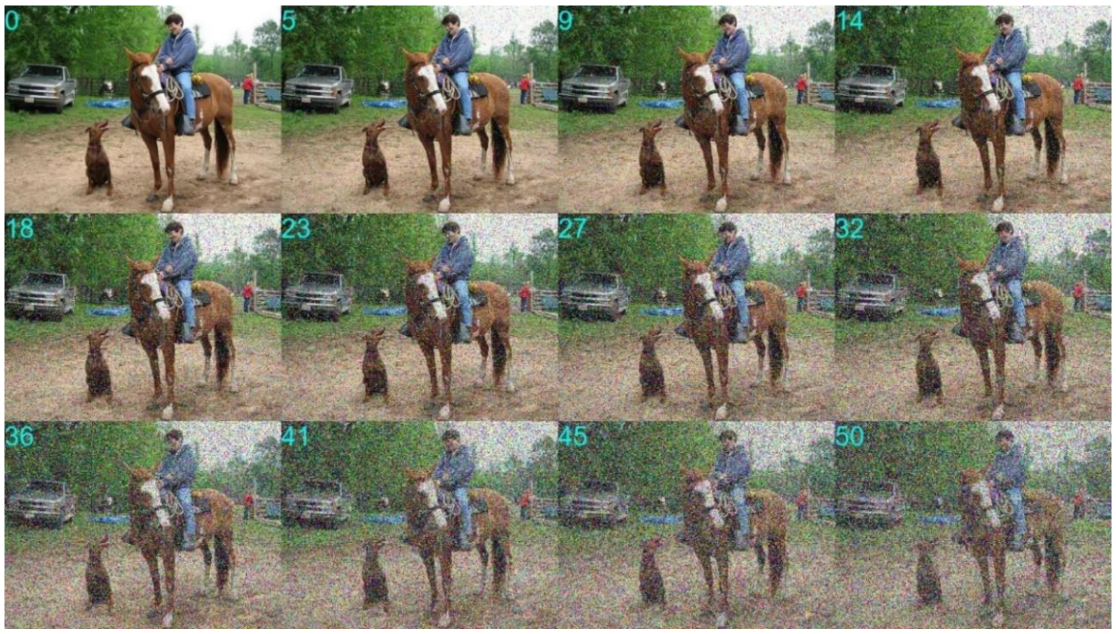


Figure 3.10 Example image displaying the various levels of salt and pepper noise, indicated in the top left in cyan as a percentage. *Salt and pepper noise levels ranging from 0% (untransformed) to 50% (very noisy). The percentage is the amount of salt and pepper to add to the image with half being "salt" or 1s and the other half being "pepper" or 0s.*

would have tested the model on all of the images at the same time, the model would have only produced one log file. This would have added significantly more post-processing time to separate each of them to be able to compare. Also, this would have taken more computational time as the model would have had to perform and store more computations in the cache from the significant increase in dataset size. This was observed when training and testing on multiple different dataset sizes (custom dataset, VOC2007). These results are discussed in the next chapter.

4 Experimental Results and Discussion

This chapter is split into three parts: training the model on the custom dataset, performing the sensitivity analysis on the VOC2007 dataset, and discussion of the overall results. The first section covers the different approaches we took on experimenting with our custom dataset. The second section includes a demonstration of how the model performs when trained on our dataset combined with the original VOC2007 dataset. In the third section, we examine the results of applying the various degradation techniques on the VOC2007 testing set and the effects on the model. The last section is the discussion, which includes comparisons and relationships amongst all results.

The metrics used in this chapter for comparison purposes are the average precision (AP) score and the mean average precision (mAP) score. Both of these scores are a combination of using the precision and recall scores of a model. Precision is how accurate (or precise) the model predictions are, while recall is the percentage of correctly detected results over all of the correct results. The AP score is calculated by taking the average of the maximum precision values at each recall value. Then, the mAP score is the mean of all of the different AP scores for each object class within the model.

4.1 Results from Training the Object Detection Model

This section includes the results using our binary class dataset – Aphylla – by itself and then combining Aphylla with the VOC2007 dataset. The Faster RCNN model we used in this work is trained on the ImageNet dataset and fine-tuned using the VOC2007 dataset. We decided to fine-tune the model using our dataset alone as well as fine-tuning on a combination of both datasets together. This allows us to observe how viable our dataset would be if combined with another and if the model would perform adequately still.

While modifying the Faster RCNN model to accept our custom dataset for training, we ran into multiple problems. While some were significant, we were able to solve all of them. Once we were able to get the model to correctly train using the custom dataset alone, we decided to combine the training set of our custom dataset with the training set of the VOC2007 dataset to train the model. The results of both of these training sets will be discussed in the following sections.

4.1.1 Experiments with Aphylla

Our first experiment consists of training the model using only our custom dataset – Aphylla. We used a Faster RCNN model pre-trained on the ImageNet dataset and fine-tuned the model with our custom dataset. We split our dataset into 50% training, 25% validation, and 25% testing, as previously mentioned in Chapter 2, which is 7370 training, 3685 validation, and 3685 testing images. Fine-tuning the model using this training set achieved 98.5% mAP score on the testing set with dragonfly as the only object class, as shown in Table 4-1. This is an astounding result for using a custom dataset for fine-tuning.

Table 4-1. Accuracy and timing results while training and testing Faster RCNN.

Dataset	Test Accuracy	Train Time (sec / epoch)	Train (FPS)	Test Time (sec / image)	Test (FPS)
VOC2007	71.1% mAP	0.824	1.2136	0.32	3.125
Aphylla	98.5% mAP	0.873	1.1455	0.291	3.4364
Aphylla + VOC2007	57.3% mAP	0.821	1.218	0.281	3.5587
Aphylla + VOC2007	96.8% AP on "dragonfly"	-	-	-	-

The first row is the baseline experiment from [46]. The second row is the results from only using the Aphylla dataset to fine-tune the model. The third and fourth rows are the results when using a combination of the Aphylla and VOC2007 datasets. The fourth row is highlighting the AP score for the "dragonfly" object class. The Faster RCNN model resizes all images to 600 pixels along the shortest size and retains the image scale. The longer side varies per image.

4.1.2 Demonstration of Combining Aphylla and VOC2007 Datasets

The next experiment we performed involves training the Faster RCNN model using a combination of our binary class dataset – Aphylla – and the multi-class dataset – VOC2007. Our intention with this demonstration is to extend the multi-class dataset by one object class by including our binary class dataset. The VOC2007 dataset contains 20 object classes and the background, so we wanted to extend this to 21 object classes by adding the "dragonfly" class from Aphylla to VOC2007. We use the split mentioned above for the custom dataset and the original split of 25% training, 25% validation, and 50% testing for the VOC2007 dataset. After combining both datasets together, we used them to fine-tune the ImageNet pre-trained weights in the model. This resulted in a 57.3% mAP score, which is much lower than the baseline result. The "dragonfly" object class scored 96.8% AP. These results are shown in Table 4-1. This is an odd mAP score given how well the model performed when training on the custom dataset alone. This prompted us to do a few more experiments and testing on both models – trained on Aphylla alone and trained on Aphylla and VOC2007 combined – to try to figure out why this happened.

4.1.3 Determining the Cause of Disparity in mAP Scores

After testing both models on various example images, we concluded that both models were overfitted on the custom dataset. When a model overfits the data, it means the model has learned the given data so much that it cannot generalize well to any other images. This means the custom dataset model generalizes to other images not containing the "dragonfly" object class very poorly by detecting dragonflies if the features remotely resemble a "dragonfly". Similarly, the model trained on both datasets over-trained on the "dragonfly" images, which resulted in the majority of the objects being detected as "dragonfly" instead of their actual object class.

After discovering the overfitting, we went back through our methodology to find the mistake we made. This error was caused by not adjusting the sizes of the datasets before combining them. Our custom dataset is 7370 training, 3685 validation, and 3685 testing images. The VOC2007 dataset is 2501 training, 2510 validation, and 4952 testing images. The VOC2007 dataset contains 20 object classes while our dataset contains only one object class. This leads to the combination dataset being very heavily weighted towards the "dragonfly" class versus all other classes. That is why the combination model is overfitted. The custom dataset model works as it should even though it is overfitted, because the ultimate goal beyond that portion of the thesis work is to be able to count the number of detected dragonflies within the images. The images on which the model would be tested are images that only contain the "dragonfly" class, similar to the dataset it was trained on. The next step after counting the number of dragonflies would be to classify each species within the images.

In the future, we would like to be able to fix the combination model so the training dataset would not be heavily skewed towards one single class. We would even out the custom dataset to make the size comparable to the VOC2007 dataset, which would create a non-overfitted model. A more detailed explanation of how we would fix this is in Section 5.3.

4.2 Sensitivity Analysis Results

We tested the model on many different ranges for each technique trying to find the perfect combination to reach the criteria mentioned in Chapter 3 – contain all critical points in the range and significant visual differences between each point. We started creating the ranges for each degradation function with 30 points, and we determined this was too computationally expensive. This would have taken multiple days to test the network a single time since there are nine functions being applied to 512 images with 30 points in each range. This totals to be 138,240 images to perform model inference on. So, we lowered the range down to 16 points, which was still too computationally expensive with 73,728 images. Ultimately, we decided on using 12 points, because this lowered the number of images down to 55,296, which is 2.5 times smaller than the original range size. This allowed us to still have an adequate number of images to determine if the test images were visually different enough to meet the criteria.

After we determined how many points would be in the ranges, we started to apply generic ranges for each of the functions onto the test images, but these were all visually inadequate. After the initial testing, we decided including the extremum was the simpler criterion to achieve first. We broke the majority of the ranges up into two parts to be able to include the original image (usually close to the middle of the range) as well as both extremes. Once we met the first part of the criteria, we had to fine-tune the parameters to meet the second criterion, which was being visually different. This part took quite a bit of testing and visualizing the test images to see if it would work properly. Note: we only made sure the test images had significant visual differences. After making sure they met the criteria, we proceeded to generalize this difference to the rest of the dataset and assume all images would be at least marginally different visually. Also, the whole dataset would be guaranteed to be mathematically different at each iteration of each function given the equations mentioned in Chapter 3.

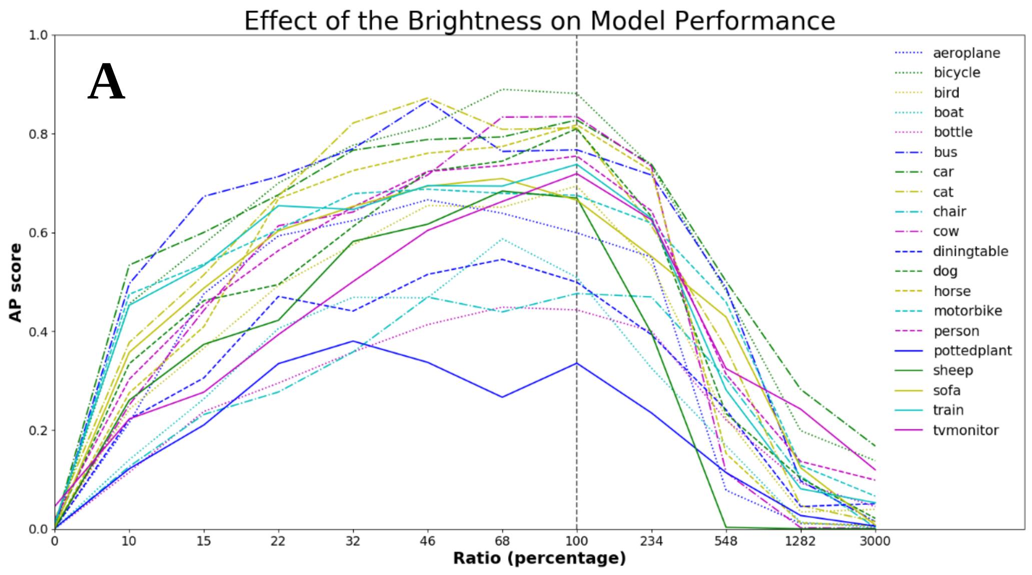
Once we determined the lengths of the ranges and met the criteria, we proceeded to test the model on all 55,296 images. This took roughly 10 hours to complete the 512 test images, 9 degradation functions, and 12 iterations of each function. For benchmark purposes, we tested the model on a single set of 512 degraded images. This took 3 minutes on average to complete. Extrapolating that average time out to all 12 functions and 9 levels of degradation each, this would come out to be 5.4 hours. This means there is about 4.6 hours' worth of overhead, which can be most likely explained by the memory usage of running 55,296 images without clearing memory between each run. In the future, we would like to be able to minimize this overhead to maximize efficiency.

4.2.1 Brightness Results

The first function we applied to the dataset for testing the robustness of the model can be found in Equations 3.2, which changes the level of brightness in the images. The range used for this technique starts at 0% (all black) and ends at 3000% (white-washed), making sure to include 100% (untransformed), as shown in Figure 4.1. While keeping our previously mentioned criteria in mind –including the midpoint and the extrema in the range and being visually different from image to image – we split the range into two sections, before and after 100% with both sections using the logarithmic space. This allows us to guarantee the range would include the original image at 100% while also extending to both extrema. This also means that we are not confined to having the same number of points before and after 100%, which allows us to have more points where the images are the most visually different from one another. The range contains the points 0%, 10%, 15%, 22%, 32%, 46%, 68%, 100%, 234%, 548%, 1282%, and 3000%.

The drawback of splitting the range into two sections is the results are difficult to coherently display on a graph, as shown in Figure 4.2. Graph (A) shows the results in the linear space, which is the hardest graph to read of the three since the majority of the points are too close together at the beginning of the graph. This graph also does not make sense to use for the brightness since both sections of the range are in the log space, which will also hold true for either section of the range for the later functions. The second graph, (B), shows the results in the log space, which is slightly clearer than the other two with the exception of the starting point being 0 which is not in the log space since $log(0)$ is undefined. Graph (C) shows the results with equidistance spacing between the points which is not proportional to the actual spacing between values. Although having this artifact, we are choosing to use this spacing because the ranges of the later functions consist of a mixture of linear and log space and we want to keep the graphs consistent between one another. This graph is also the easiest to read of the three and objectively more aesthetically pleasing to the eye. Based on these conclusions, we have chosen to use the spacing of graph (A) on all subsequent result graphs to maintain consistency when presenting the results of each function.

Starting with the pitch black images (0% brightness), the majority of the AP scores for each object class are 0, which means no detections could be made. The outlier at 0% is the "TV monitor" class



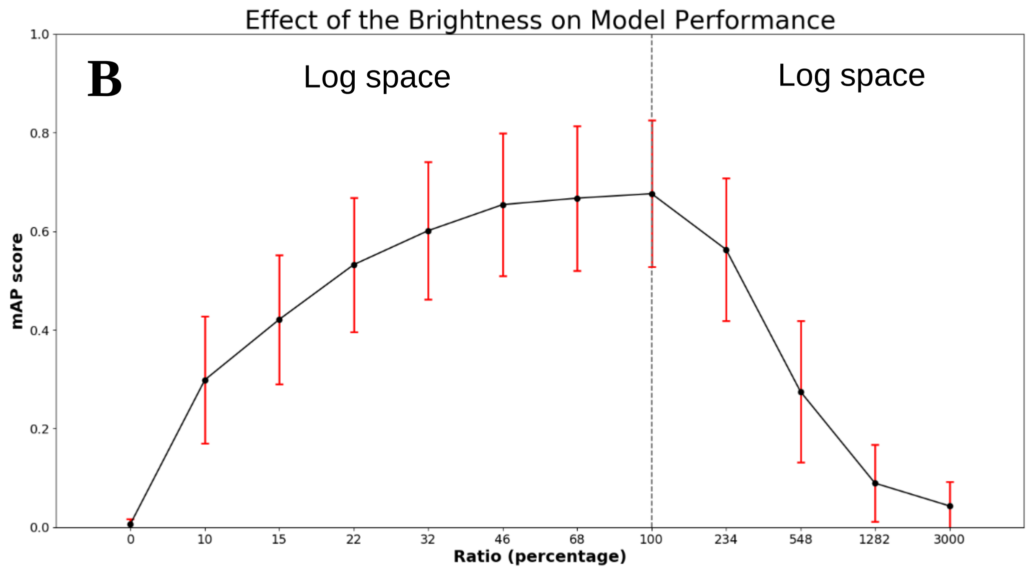


Figure 4.1 Effect of image brightness on model performance: (A) AP by VOC class and (B) mAP overall. *Model performance was measured for images ranging in brightness from 0% (all black) to 3000% (white-washed), including 100% (untransformed, vertical dashed line). As the brightness increases from 0% to 10%, performance increases rapidly since the model is able to distinguish features even before objects become visible to a human viewer, due to the image normalization pre-processing used by the model. From 10% to 100%, performance increase steadily as object features become more visible. Above 100%, model performance decreases markedly as images become white-washed. Note: On (B), log space on both sides of 100% indicates two different log spaces scales being used.*

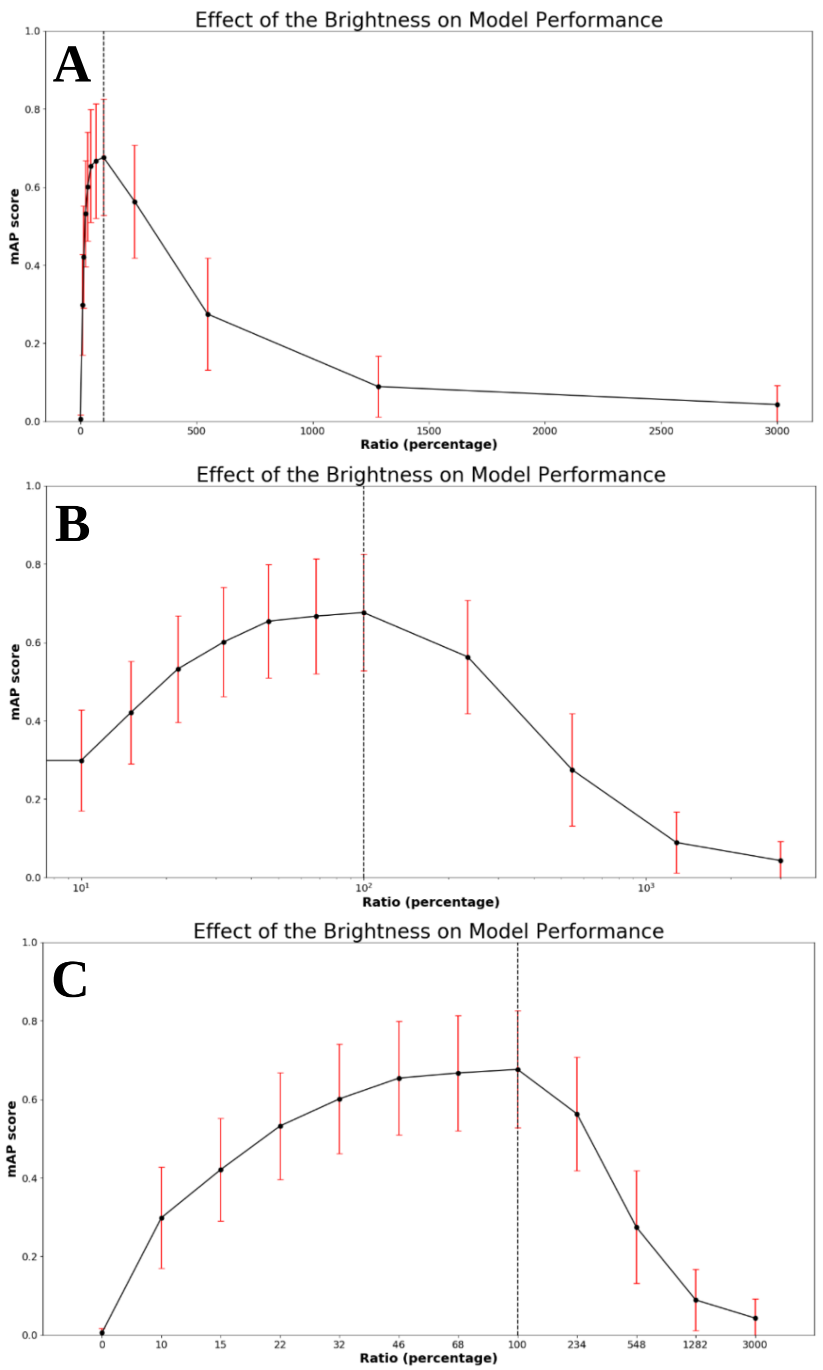


Figure 4.2 Three different graphs displaying the brightness results using (A) linear space, (B) log space, (C) equidistant spacing between points. *The vertical dashed line is the untransformed image in each graph. The first graph is very difficult to read because all of the points are too close together at the beginning. The log space graph is a better representation of the data except 0 is not on the log space graph, which is the first point in the range, and this graph would not be feasible for the later functions since they use a mixture of linear space and log space. The last graph is the easiest to read even though the spacing is not proportional to the distance between each point.*

(0.0456 AP score). This can be explained because TV monitors (when turned off) are mostly black rectangles, which means the deep learning model may detect a false positive with a very low chance anywhere in the image since it is all black. Over the next few points, from 0% to 100%, the curve follows a very sharp increase and tapers off when approaching 100% as shown in Figure 4.2a. This increase is occurring as the objects become steadily more visible. Once the images exceed 100% brightness, the mAP score rapidly decreases because the images are becoming increasingly white-washed, which in turn make the objects harder to detect.

There are a few oddities within the results of this function. The first one is the “potted plant” class, which is the main outlier. This object class performs the worst in the majority of our experiments, including the initial VOC experiment [46]. This is caused by the training set not having enough images of this class (5.3% of the training set [46]) for the model to perform adequately on this object. The next peculiarity is a few of the object class detections start outperforming the original image before the images reach 50% brightness. This could be explained by the darker (lower brightness) images improving the distinguishability between the objects. This helps the model more easily detect those objects because the object textures are slightly enhanced, even though the overall images are not optimally displayed for a human viewer. Another interesting result is the “car” object class, which seems to perform the best towards both ends of the range. This occurrence happens because the “car” object class is the second largest class in the VOC dataset with “person” being the highest [46]. The reasoning behind “car” performing better than “person” at the ends of the range is because “car” objects have a single shape, whereas “person” objects can be various shapes.

Adjusting the brightness is among the most fundamentally important functions for testing the robustness of the model since brightness is one of the core attributes of an image along with the saturation and contrast. The results from applying the brightness adjustments meet our expectations almost completely: the model performing well between 50% and 200% brightness and the performance sharply decreasing outside of this range. This means the model is robust enough to detect objects in images where the lighting is not as perfect as the majority of computer vision images. Corrective actions will have to be taken outside of this range, which would include pre-processing steps performed on the images before deploying the deep learning model on the imagery. These steps would include lowering or raising the brightness based on the results and the image.

In conclusion, varying the levels of brightness within the test images proved to be a useful technique in determining robustness. The results showed the range between 50% and 200% as performing the best. This is intuitively expected since having ideal image intensity is a key factor for model robustness.

4.2.2 Saturation Results

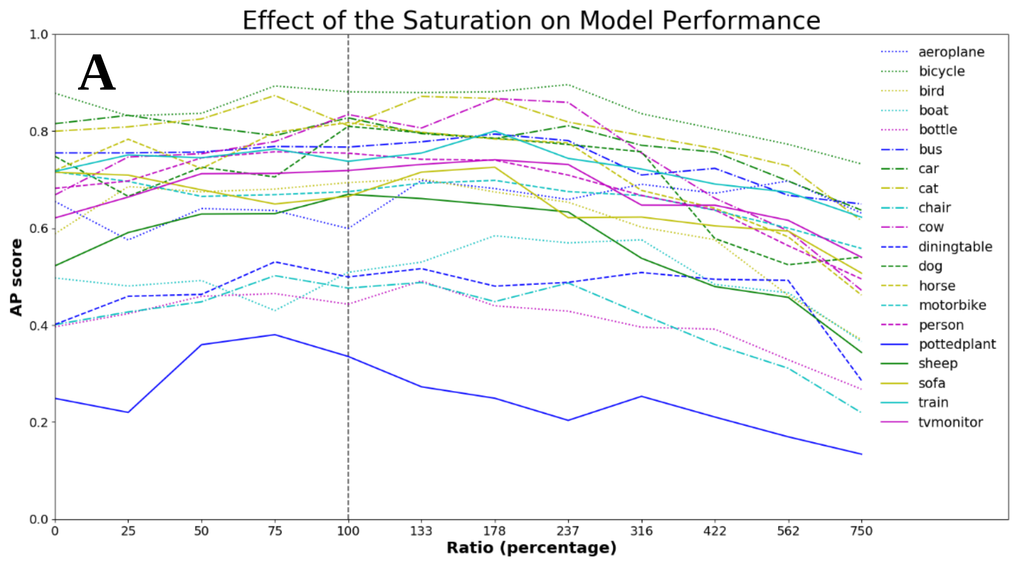
The next method we used for testing the model is applying the saturation adjustment equations to the testing images, which can be found in Equations 3.3 and 3.4. This was a difficult method to determine the range to meet our criteria while obtaining good results simultaneously. After testing many ranges and not being able to see a significant difference between the images for the majority of them, the one we ultimately decided to use starts at 0% (grayscale) and goes to 750% (heavily saturated) making sure to contain 100% (untransformed), as shown in Figure 4.3. We split the range into two sections for the same reasons as we did previously – to maximize the visual difference between images and to make sure to capture the three critical points. For the section before 100%, we used linear spacing by increasing the amount of saturation by 25% until 100%, creating a total of five points. After 100%, we used the logarithmic space from 100% to 750%, which created the last seven points. The points contained in this range are 0%, 25%, 50%, 75%, 100%, 133%, 178%, 237%, 316%, 422%, 562%, and 750%.

As shown in Figure 4.3b, the mAP score of the results start slightly lower at 0% (0.627) than they are at 100% (0.677). This is because the model does not rely solely on the color features of the objects but on a combination of the color and texture features. Past 100% saturation, the highest mAP score is 0.686 at 133%. This is because the colors in the images are easier to differentiate from one another than in the original images. This marginally boosts the mAP scores since the model can more easily detect the objects. After 133% saturation, the mAP scores steadily decrease down to 0.473 at the last point in the range (750%). At this point, the objects are much less distinguishable since all the reds and greens are overly exaggerated and mixing together between similarly colored objects.

Similar to the brightness adjustment, the “potted plant” object class performs very poorly compared to the other classes. This result is unique because “potted plant” is lower than any other class at every point in the graph. This can be explained using the same explanation as before – the object class is under-represented in the VOC training set at only 5.3% [46]. Another interesting result is the “bicycle” object class, because it performs the same or better than all of the other classes at each different saturation level. The reasoning behind this is a bicycle is a rigid object. It almost never changes shape, which would also imply the texture features would generally stay the same for each “bicycle” object. This allows the model to almost always detect bicycles within saturated images since only the background behind the bicycle can change within the detected bounding box. The rest of the results fall within the expected range for each point.

Varying the levels of saturation in the test images is another fundamental way to test the robustness of the model. This is because the saturation is one of the components of image intensity along with brightness and contrast. These three attributes are the main controllers of how an image is visually perceived, giving the model a good benchmark for testing to determine the performance. The range where the model performs the best on saturated images is between 50% and 200% saturation, same as the brightness. The mAP results stay above 65% within this range, which means the model will be able to detect the majority of objects even though the saturation is not perfect. Corrective actions need to be taken outside of this image saturation range, which should include raising or lowering the saturation given the initial results and visual cues within the images.

Overall, saturation plays a lesser role than the other image intensity factors in determining how robust the model is. This is intuitively not expected because brightness and contrast both contribute heavily to robustness, which are the other two main attributes on image intensity.



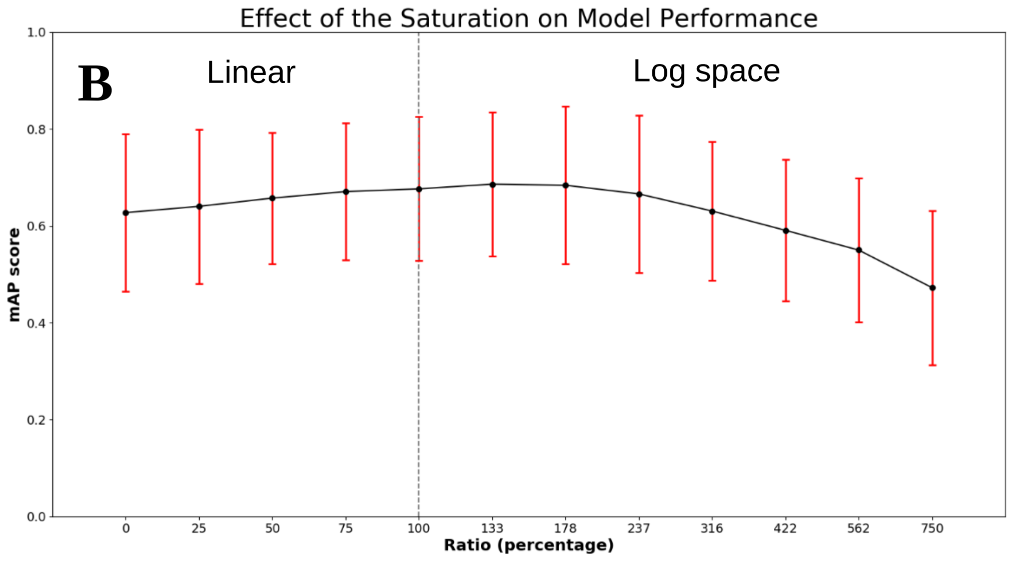


Figure 4.3 Effect of image saturation on model performance: (A) AP by VOC class and (B) mAP overall. *Model performance was measured for images ranging in saturation from 0% (grayscale) to 750% (heavily saturated), including 100% (untransformed, vertical dashed line). As the saturation increases from 0% to 133%, performance increases gradually indicating the model does not rely solely on color features. Above 133%, model performance decreases steadily as images become saturated.*

4.2.3 Contrast Results

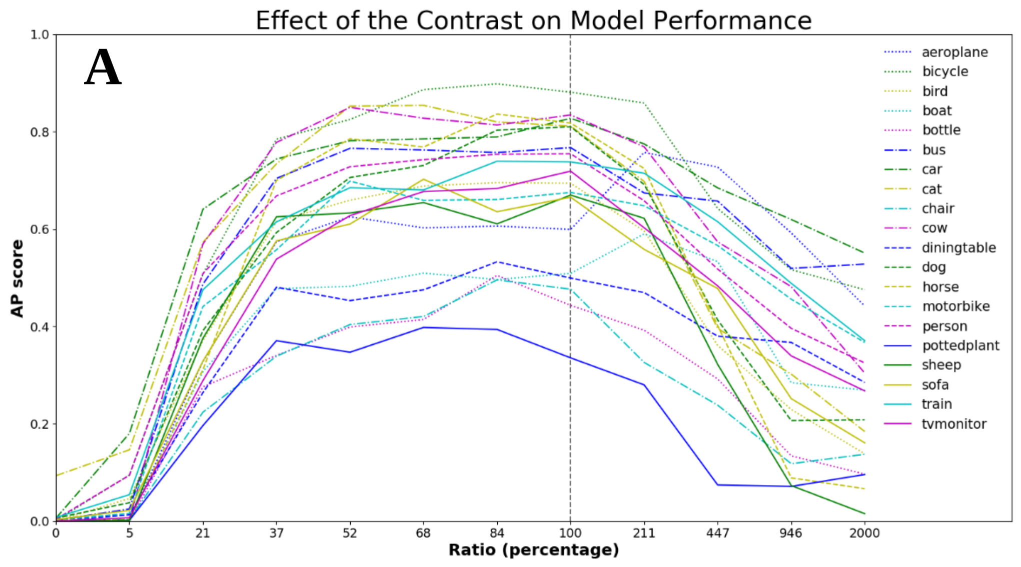
The third attribute of image intensity is contrast. We applied Equations 3.5, 3.6, and 3.7 to the test images for adjusting the contrast. The range we decided to use starts at 0% (all gray) increasing to 2000% (heavily contrasted), while making sure to include 100% (untransformed). Using the same criteria as before, we had to split the range into two sections again. Before 100%, we used the linear space to determine the points, which comprises of seven points. The point at 0% was included into the range after we created the initial range to make sure the low-end of the spectrum was covered. The reasoning for this is because starting the linear spacing at 5% and going to 100% was clearer to see the visual differences than starting at 0%, as shown in Figure 3.4. The second reason for doing it this way is from 0% to 5% contrast is already significantly visually different. The second section of the range uses the logarithmic space for determining the points. This allows this section to be able to show higher values of contrast and still maintain the needed 12 points within the range. The range consists of the points 0%, 5%, 21%, 37%, 52%, 68%, 84%, 100%, 211%, 447%, 946%, and 2000%.

The model performance was abysmal from 0% contrast to 5%, as shown in Figure 4.4, which is to be expected since the objects are almost indistinguishable from one another on account of how gray the image is. From 5% to 37%, there was a steep increase in the mAP scores explained by the objects in the images becoming more distinguishable, but not fully yet. After that, the results become much more stable until 211% where the results start sharply declining. Within this range, the objects are at the best contrast for the model to detect, as shown in Figure 4.4b, with 100% contrast giving the best detections.

There are a couple irregularities within the contrast results, namely the “cat” and “potted plant” object classes. We will not go in-depth on the “potted plant” class since the explanation for why it is gives a lower than usual result holds true for each function – the training set of images only contains 5.3% of the “potted plant” class in the entire set [46]. As for the “cat” object class, it has a 0.09 AP score at 0% contrast, which is within the range of random guessing for 20 classes. The most reasonable explanation for this artifact in the results is the same reasoning as for the “potted plant” class, the training set only contains 6.5% of the “cat” object class [46]. There is also a lower number of “cat” objects in the training set than there are “potted plant” objects, 186 cats vs 248 potted plants.

Contrast is one of the core characteristics of image intensity, which is why adjusting the levels of contrast in the testing images is a crucial function in testing model robustness. The contrast results adhere to the same overall curvature of the brightness results since both are directly related to image intensity. The section of the range where the results are the most optimal is from about 50% to 200% contrast, which falls in line with the optimal ranges for both brightness and saturation. Brightness, saturation, and contrast are the main contributors to image intensity, which validates why the optimal ranges for all three coincide with each other. Outside of this range, pre-processing steps would need to be taken on the testing images to equalize the contrast based on the results from the model.

In summary, contrast performs similarly to brightness (Section 4.2.1), which are both key factors in image intensity. This means adjusting the contrast is also an essential method for determining



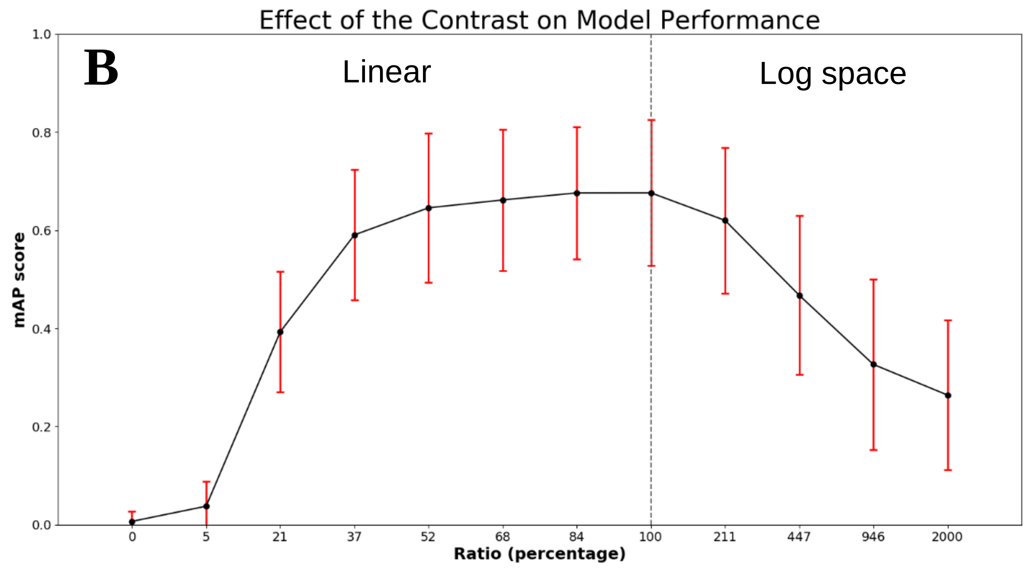


Figure 4.4 Effect of image contrast on model performance: (A) AP by VOC class and (B) mAP overall. *Model performance was measured for images ranging in contrast from 0% (all gray) to 2000% (heavily contrasted), including 100% (untransformed, vertical dashed line). At 0% contrast, the class "cat" stands out as unusual having an AP score of 0.09, which is within the range of random guessing with 20 classes. As the contrast increases from 0% to 5%, performance only slightly increases since the model is not able to distinguish features from the low contrasted images. From 5% to 21%, performance sharply increases as the model is able to differentiate objects from one another. Above 21%, model performance behaves analogous to brightness (fig. 10), due to both being related to image intensity.*

robustness. This is intuitively expected because image intensity is a large influence in deep learning models.

4.2.4 Gaussian Blur Results

Another method we used to create non-pristine images is applying Gaussian blur to the test images using Equation 3.8. This function allowed us to simulate blurry, out-of-focus images to test how robust the model is to this situation. After initially testing a range that used linear spacing from 0 to 12, we determined that this would not fill the visually different criteria for the upper end of the range. Also, in the lower end of the range, going from 0 to 1 pixel blur radius was too large of a difference. This led us to split the range into two sections, 0 (original image) to 2 in increments of0.5 and 2 to 9 in increments of 1. This keeps the range spanning 12 points, and each iteration is much more visually different than the aforementioned range. The points in the range (all in pixels for the blur radius) are 0, 0.5, 1, 1.5, 2, 3, 4, 5, 6, 7, 8, and 9.

The results after applying the varying amounts of Gaussian blur to the test images are displayed in Figure 4.5. The mAP curve in Figure 4.5b follows our intuition of what the results should be. Intuitively, the performance of the model should have started at the maximum AP scores for each object since the first point in the range is the original image. After that, it should degrade marginally as the images become slightly blurry. Then, the performance should sharply decrease until the image is completely out of focus. Finally, the end of the curve should continue steadily decreasing until the AP scores of each object reach almost 0. The actual results follow our intuition completely.

As per usual, the “potted plant” object class performs the worst of the classes. Another irregularity is the “person” class is detected the best at the end of the range, and it stays stable around 0.10 AP score. This is an odd result since 0.10 AP is higher than expected given that the image is almost completely blurred by the ending of the range. The only plausible explanation for this is the “person” object class is the best trained object class in the model. The VOC training set is composed of 41% of the images containing one or more of the “person” object class [46]. Also, 37.4% of all of the objects in the dataset are “person” objects. For perspective, the next largest object class is “car” at 15% of the dataset. Given how large the “person” class is, this explains why the AP score is abnormally high given that the images at the end of the range are not completely blurred.

One of the key factors in testing the robustness of the model is testing how well it will perform on blurry images. This is because blurry or out-of-focus images are a typical type of non-pristine image to test an object recognition model on. Based on the results, the model performance starts heavily degrading after a 1 pixel blur radius. This is where the edges of the objects start becoming hard to distinguish for the model even though the objects themselves are easy to distinguish with the human eye. This means the acceptable range of usage is trivial. The recommended corrective actions to be taken when needing to detect objects in blurry imagery is to apply image sharpening.

In short, applying Gaussian blur to the test images provides great insight into model robustness,

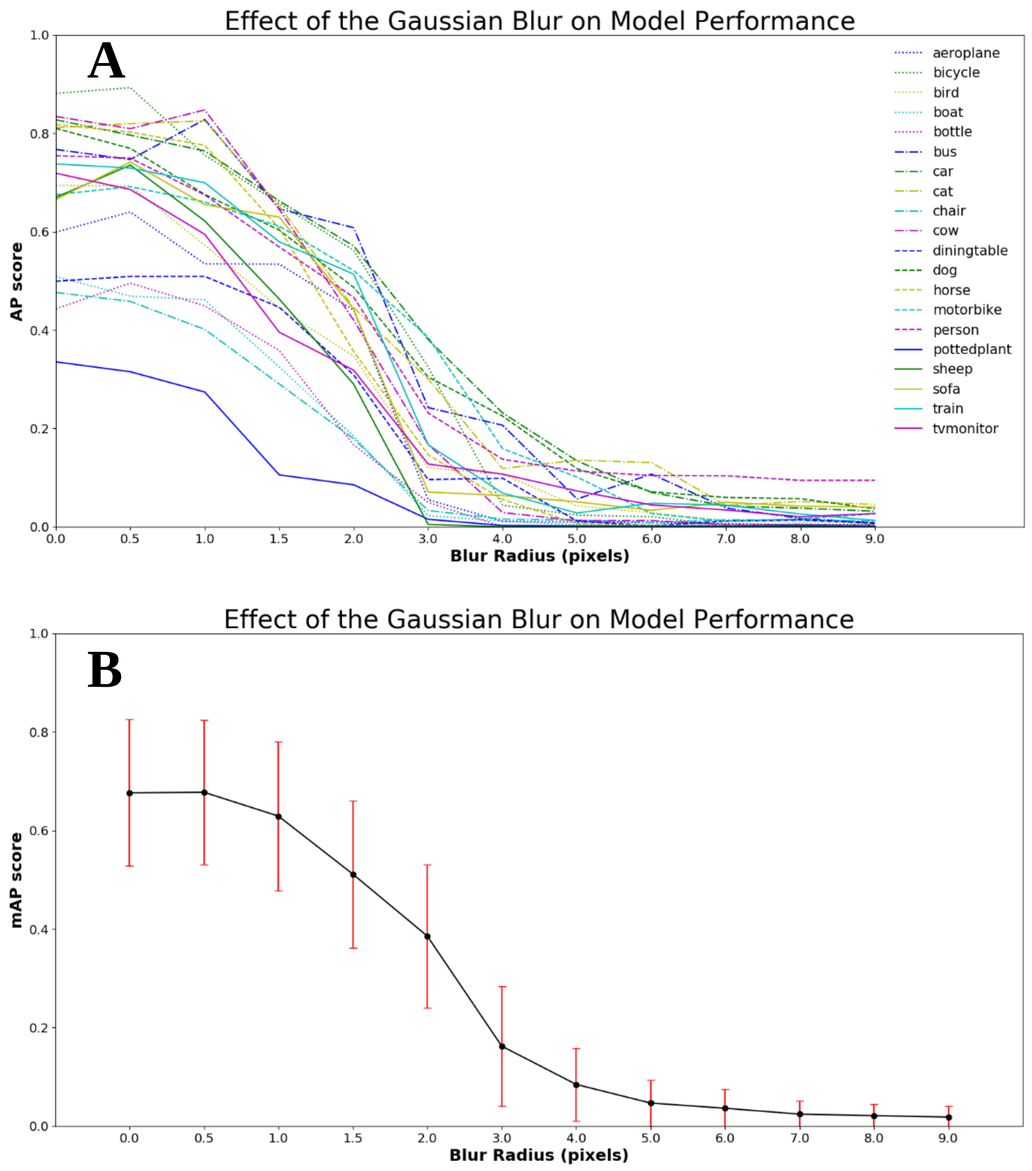


Figure 4.5 Effect of Gaussian Blur on model performance: (A) AP by VOC class and (B) mAP overall. *Model performance was measured for images applied with Gaussian blur with radii ranging from 0 (untransformed) to 9 pixels (heavily blurred). As the radius increases from 0 to 0.5 pixels, overall performance stays the same since applying a Gaussian blur with radius of 0.5 pixels is analogous to anti-aliasing. From 0.5 pixel radius to 4 pixels, performance sharply decreases as features become less visible from the blurring. After 4 pixels, model performance steadily decreases to nearly 0.00 AP score, which indicates the object features are becoming unrecognizable.*

showing how quickly the performance degrades as the blurriness increases. This result is intuitively expected, since the objects become harder for the model to detect within very few steps.

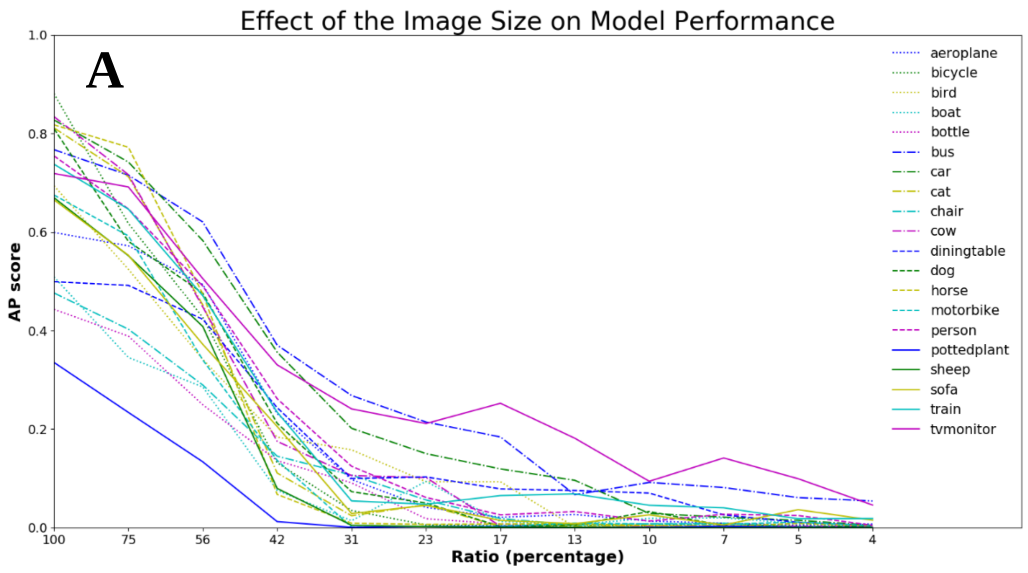
4.2.5 Image Size Results

Another method we chose to apply as a pre-processing step while testing for robustness is resizing the images to simulate poor resolution. The function we used to accomplish this can be found in [56]. The range we chose for this technique starts at 100% resolution and decreases to 4% using the logarithmic space, as shown in Figure 4.6. The way we implemented the range emphasizes the lower percentages more than the higher. This is because the lower percentages show a much larger visual difference between the images, which is our first criterion for the range. Our second criterion was also much easier to adhere to because there are only two critical points for this method, 100% resolution and 0%. This allowed us to not have to split the range into two sections. The points within the range are 100%, 75%, 56%, 42%, 31%, 23%, 17%, 13%, 10%, 7%, 5%, and 4% image resolution.

The results of this function, shown in Figure 4.6, uniformly decrease as image resolution decreases. Beginning with the original resolution image, the mAP score starts rapidly decreasing until 42%, at which point, the results slow to a steady decrease. These results show that the model is not able to detect objects adequately when there is a large amount of pixelation in the image. The mAP score starts at 0.677 at 100% resolution, and at the next point (75%), the mAP drops to 0.575. This is a very noticeable decrease in the amounts of correct detections. When the range reaches 42% resolution, the mAP score is significantly lower at 0.185. After this position in the range, the mAP score gradually decreases to 0.009 by the last point in the range (4%). By the end, the objects are also entirely indistinguishable given how low resolution the images are.

There are a couple anomalies in these results, which are similar to the previous techniques. The typical one is the “potted plant” object class. This class has scored marginally lower than all of the other classes in the majority of the techniques so far, and for the same reason – it is under-represented in the VOC dataset [46]. Another object class with a greater difference than the rest is the “tv monitor” object class, which has been mentioned in Section 4.2.1. Towards the end of the range, this class has a higher AP score than most of the others. This has a similar explanation as the brightness results as well – a tv monitor is mostly a black/gray rectangle when turned off, so the model is detecting the lower resolution images as containing the “tv monitor” object class when the enlarged pixels are darker than the other pixels in the images. An example of these enlarged pixels is shown in Figure 3.6. The remaining results fall within the expected range for each point.

Varying the pixelation within the images gives us insight into how the model detects objects. This understanding is given by showing us the relationship between how pixelated or low resolution an image is versus how well the objects are being detected. As the pixelation increases, the object detection rate markedly decreases. This occurs because a deep learning model is trained on three key features of an object (as well as many other features): the texture, outline, and color of the object. The object texture vanishes as soon as the resolution of the image starts to decrease.



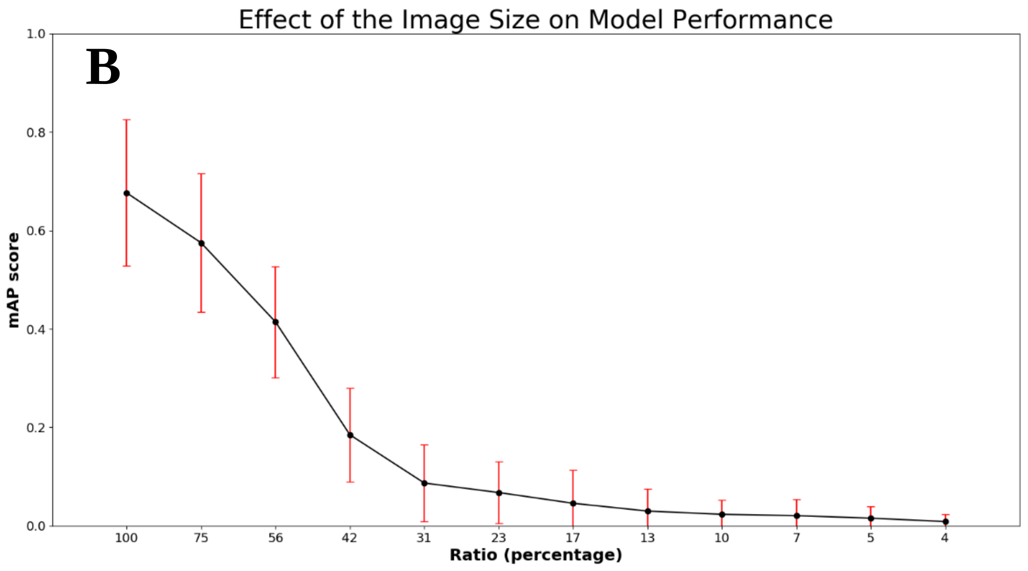


Figure 4.6 Effect of image size on model performance: (A) AP by VOC class and (B) mAP overall. *Model performance was measured for images ranging in size from 100% (untransformed) to 4% (extremely pixelated), which were resized back to the original size to replicate pixelated images. As the image size decreases from 100% to 42%, performance decreases markedly due to the model not being able to distinguish features even though objects are still easily detectable by a human viewer. Below 42%, model performance decreases steadily as the images become more pixelated.*

Similarly, the outline of the object starts to get pixelated, which in turn blurs the edges of the object. The color features stay intact until the lower levels of the image resizing, which reinforces the idea mentioned in Section 4.2.2 – deep learning models do not rely solely on color features, but rather, they rely on a combination of the object features. The acceptable amount of pixelation in the test images is almost too minuscule to mention given the model does not perform well after the original image size. Corrective actions would need to be taken on pixelated imagery, but at this point in time, we are not sure on how to go about fixing pixelation. There are only methods to help mitigate pixelated images such as applying a small Gaussian blur filter or applying a sharpening filter.

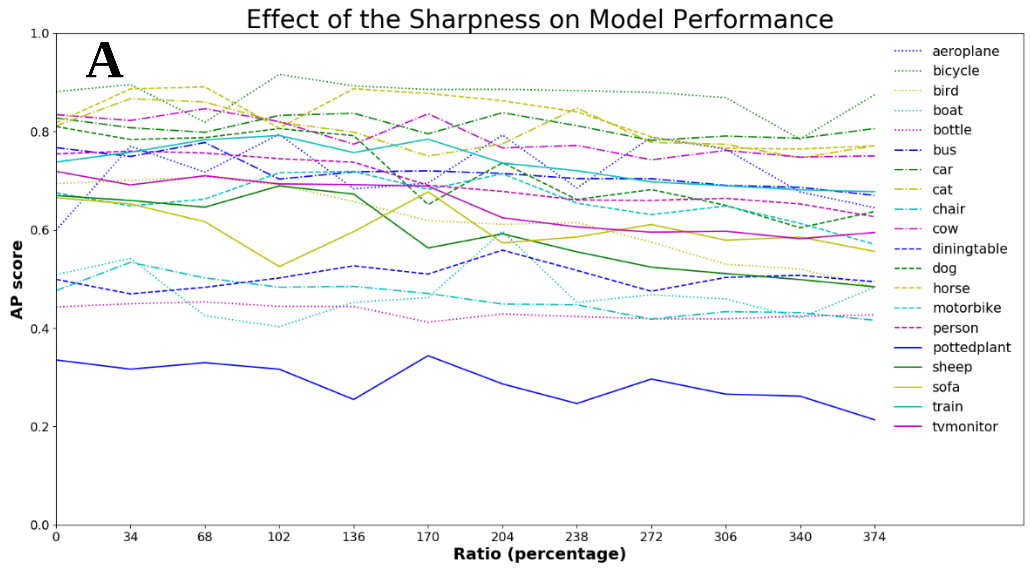
In conclusion, testing the model on low resolution imagery is essential for assessing model robustness since this type of degradation is typical in natural images. This method shows how the performance decreases as the pixelation increases. The overall result is intuitively expected because the object features become indistinguishable to the model quickly.

4.2.6 Sharpness Results

The next function we applied to the testing set is image sharpness adjustment using Equation 3.9. This technique uses the Gaussian blur equation (Equation 3.8) and subtracts that from the original image. This enhances the object edges within the image, making them sharper. This was the most difficult technique to find a suitable range for. We created and tested many ranges and still could not get substantial results. All of this was using multiple different spacings in both the linear and the logarithmic space. We also could not get them to meet our two criteria. So, we decided to keep this range simple. The final range starts at 0% sharpness (untransformed) and increases linearly to 374% (very sharp) in increments of 34%. This allowed us to maintain a very loose interpretation of our criteria – visually different and including the critical points. The points in this range are 0%, 34%, 68%, 102%, 136%, 170%, 204%, 238%, 272%, 306%, 340%, and 374%, as shown in Figure 4.7.

The mAP at 34% sharpness (0.6883) is slightly higher than the mAP at 0% (0.6765), as shown in Figure 4.7b. This is because increasing the image sharpness enhances all of the edges within the image. This makes the objects slightly more distinct, which in turn allows the model to detect the objects easier. After 34% sharpness, the results become almost completely linear, decreasing at a rate of 0.0082 mAP per 34% increment. In other words, the model performance decreased by 0.091 mAP by the end of the range. This means the mAP for the model decreases at an extremely low rate while increasing the sharpness.

Similar to the saturation results in Section 4.2.2, the "bicycle" object class scores the same or higher for all points except at 68% sharpness. The reason for this is explained in Section 4.2.2 – the "bicycle" object class is typically a rigid object. The other notable result is the "potted plant" object class, which scored significantly worse at all points than any other class. The explanation for this has been mentioned in each section before this one – under-represented in the VOC dataset [46]. The rest of the results fall with the margin of error at each point.



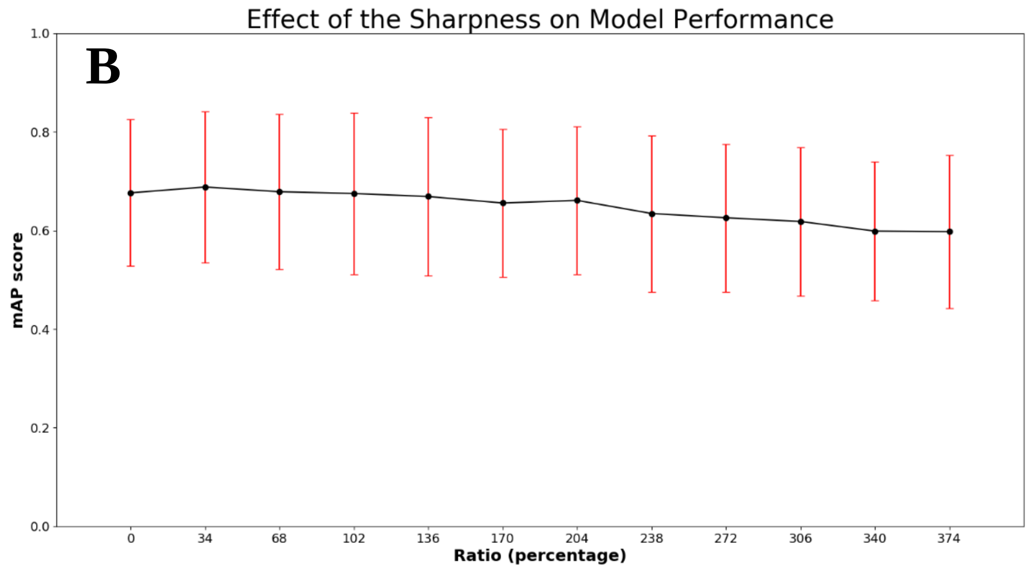


Figure 4.7 Effect of image sharpness on model performance: (A) AP by VOC class and (B) mAP overall. *Model performance was measured for images ranging in sharpness from 0% (untransformed) to 374% (very sharp). For the entire range of the sharpness, model performance stays the same within the margin of error with a slight decrease in performance with increased sharpness. This can be explained by the sharp images not being too different from the original images. The only thing that gets amplified in this operation is the edges become more visible, and the color stays the same, which are the two key features the model relies on.*

Image sharpness is related to image intensity comparable to brightness, saturation, and contrast, although sharpness is not as tightly bound to the intensity as the others. This is why the results are much less similar. Also, varying the levels of sharpness within the test images is not an adequate assessment for model robustness, because the model is able to handle significant levels of sharpness before performing poorly. Ultimately, this means our model is adept at detecting objects in overly sharp imagery. Once the model performance drops to unacceptable levels, the corrective action that would need to be taken is to apply Gaussian blur to the images which are overly sharp. This should fix the majority of the detection issues in those images.

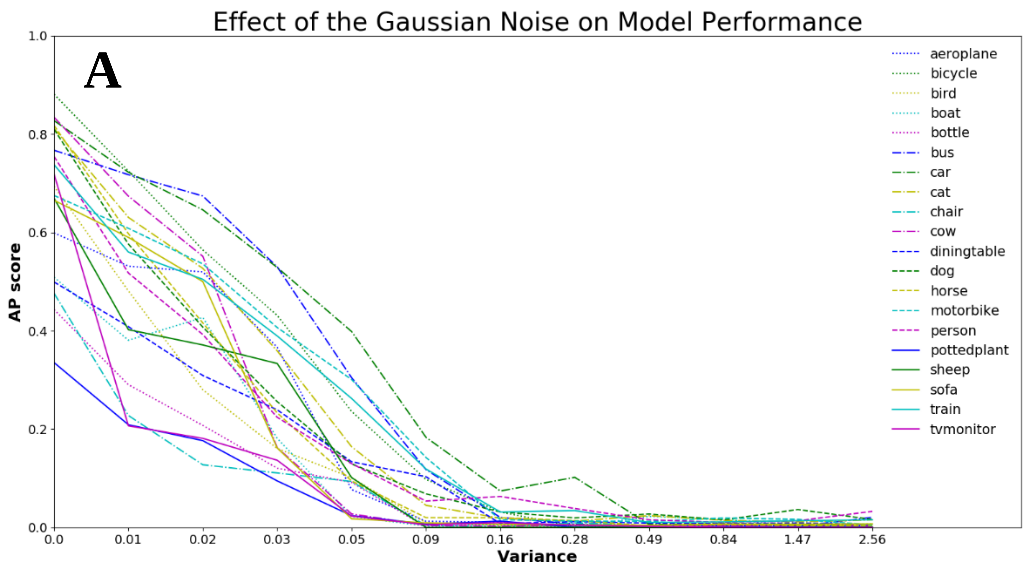
To summarize, image sharpness is not a core factor for determining model robustness since the object features do not undergo a significant change as the other degradation methods we use in our experiments. This allows the model to be robust towards all tested amounts of sharpness, which is intuitively to be expected.

4.2.7 Gaussian Noise Results

The next method we use for assessing the robustness of the model is Gaussian noise, as shown in Equations 3.10 and 3.11, which uses a similar equation to Gaussian blur. This type of noise is easy to visually detect within an image, which made the visually different part of our criteria trivial to achieve. The second part of the criteria – include the critical points within the range – was also very simple to attain since there is no midpoint, like in Section 4.2.1 with 100% brightness needing to be encompassed within the range. The critical points are only the beginning and ending of the range, or the untransformed image and the noisy image respectively. This also means we did not have to split the range into two different sections. We used the logarithmic space to create all of the points needed. The variances used for the range are 0, 0.01, 0.02, 0.03, 0.05, 0.09, 0.16, 0.28, 0.49, 0.84, 1.47, and 2.56.

As shown in Figure 4.8, the model performance rapidly declines from 0 to 0.09 variance, decreasing from 0.677 mAP to 0.05. The reasoning behind this is as small amounts of noise are introduced to the images, the features that the model relies on get distorted, which in turn quickly decrease the detection rate. After 0.09 variance, the performance continues to decrease steadily until it reaches 0.01 mAP at 2.56 variance. By this point, the images are extremely noisy even though a human viewer might still be able to detect certain objects, the model cannot.

There are not any major outliers within the results of the Gaussian noise. The ones that stand out are the “car” and “bus” object classes in the beginning and the “dog” and “person” classes towards the end of the graph in Figure 4.8a. The “car” object class consistently scores higher than most other classes from 0 to 0.49 variance. One reason behind this could be because “car” is the second largest class in the VOC dataset at 15% [46]. This means the model is trained to detected cars than most of the other object classes. Another reason could be that cars have many different appearances, for example, size, shape, and color, which would make the model more robust when adding noise to the images. The “bus” object class does not stand out as much as the “car” since it only scores higher than most classes from 0.01 to 0.05 variance. This could be explained with the same reasoning as the second portion as “car” although to a lesser extent – different appearances,



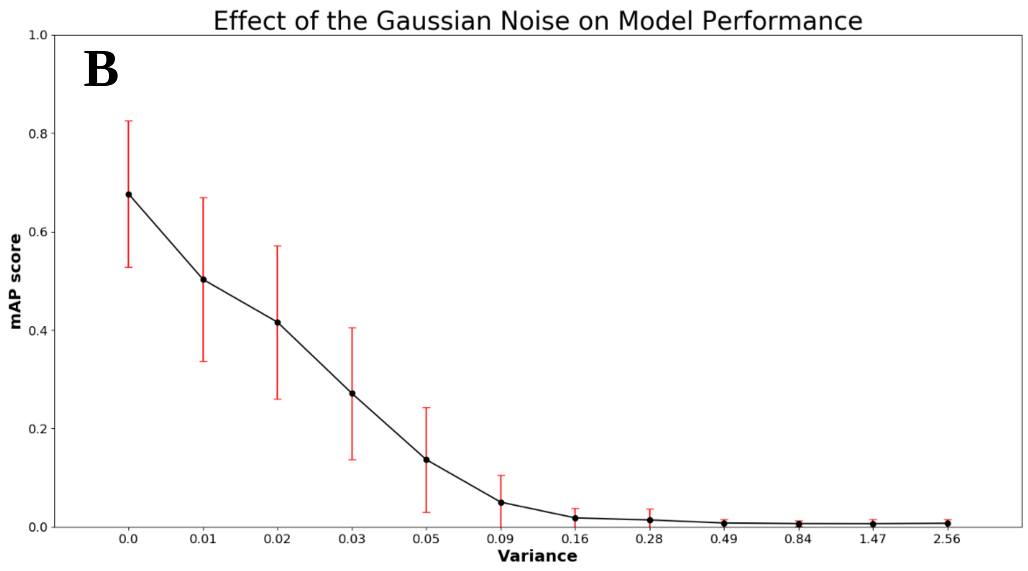


Figure 4.8 Effect of Gaussian noise on model performance: (A) AP by VOC class and (B) mAP overall. *Model performance was measured for images applied with Gaussian noise with variance ranging from 0 (untransformed) to 2.56 (very noisy). As the Gaussian noise variance increases from 0 to 0.09, performance decreases sharply due to low levels of noise distorting the object features quickly. After 0.09 variance, model performance keeps decreasing steadily as features are almost unrecognizable by the model even though objects are distinguishable by a human viewer.*

making the model slightly more robust towards detecting them. The classes at the end that score higher than the rest are the “dog” and “person” classes. They only score slightly higher than the others at 1.47 for “dog” and 2.56 for “person”. These are both within the range of randomness, which is the best explanation since both are still below 0.04 mAP.

Overall, adding Gaussian noise is a beneficial test for model robustness because it allows us to see how quickly the model performance degrades at the first signs of noise. The acceptable amount of this type of noise before the model performance significantly degrades is non-existent, since 0.01 variance decreases the mAP score from 0.677 to 0.503. There are not many corrective actions that can be applied to noisy imagery. The ones that do exist do not fix the problem well, such as adjusting the luminance or color of the image to simulate changing the exposure. Gaussian noise is an additive type of noise, whereas the next test we perform is on speckle noise, which is multiplicative.

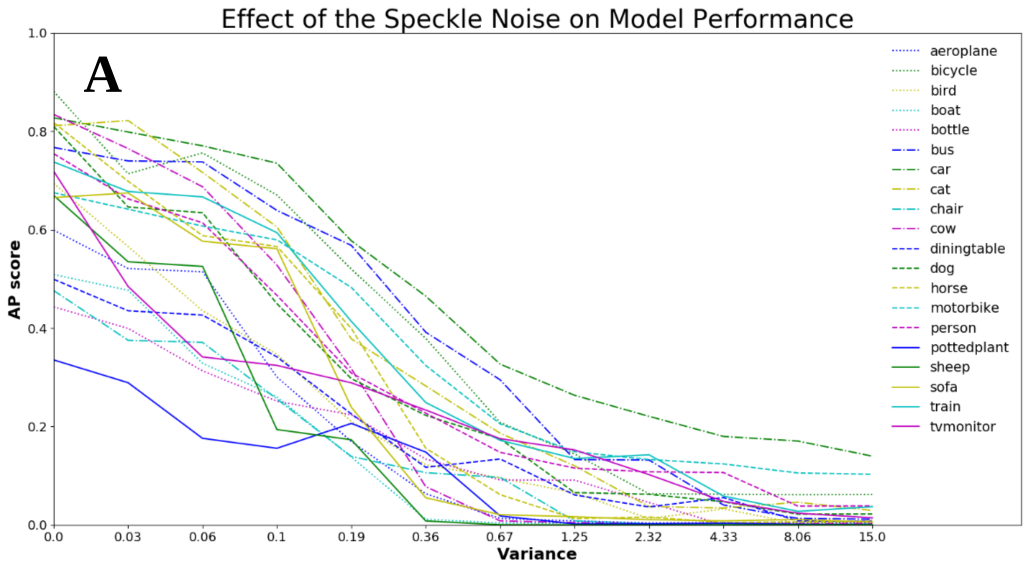
In conclusion, Gaussian noise is one of the three types of noise we use for testing the model robustness. The model performs poorly after applying this type of noise to the test images. This is intuitively expected, because adding small amounts of noise to images alters the object features significantly.

4.2.8 Speckle Noise Results

Speckle noise is the second type of noise we applied to the testing set of images using Equation 3.13. This form of noise is multiplicative as mentioned at the end of the previous section, Section 4.2.7. Gaussian noise is calculated by adding the result of Equation 3.10 into the original image, whereas speckle noise is calculated by multiplying the result of the same equation (3.10) with the original image and adding that result to the original image. The two types of noise only have subtle differences as can be expected with how similar the equations are. Another parallel between the two is how the criteria is met – noise is inherently easy to visually see within images and no midpoint needs to be included in the range. This allows the range to be one consistent string of numbers using the logarithmic space starting at 0 variance and increasing to 15, as shown in Figure 4.9. These points are 0, 0.03, 0.06, 0.1, 0.19, 0.36, 0.67, 1.25, 2.32, 4.33, 8.06, and 15 variances.

The mAP score gradually decreases as the variance increases from 0 to 0.36, as shown in Figure 4.9b. After 0.36 variance, the model performance decreases at a slower pace. The marked decrease in the first half of the range can be explained by the noise altering the object features on which the model is trained, thus, in turn, decreasing model performance very quickly. This indicates the model is not robust in terms of speckle noise, which shares a similar conclusion as Gaussian noise.

The two classes that stand out the most in the AP score results are “potted plant” and “car”. The “potted plant” object class is the class that stands out by scoring lower than the others in the majority of the different functions we have tested so far. The reasoning has also been explained multiple times – this class is under-represented in the dataset [46]. As for the second class, “car”, the AP score is higher than most of the other classes at almost the entire range. This class is the second largest object class in the VOC dataset, meaning the model has been trained to detect “car”



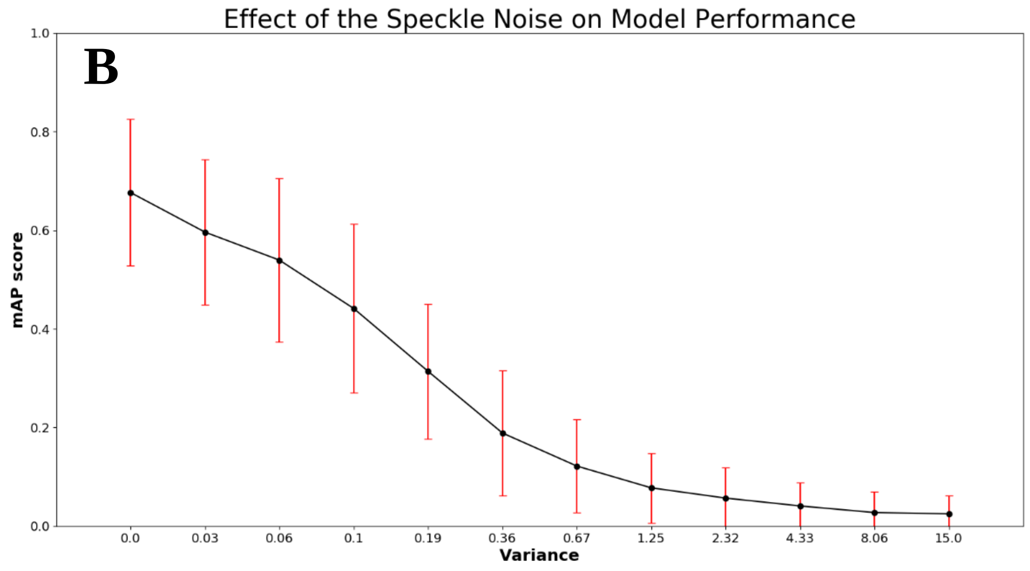


Figure 4.9 Effect of speckle noise on model performance: (A) AP by VOC class and (B) mAP overall. *Model performance was measured for images applied with speckle noise with variance ranging from 0 (untransformed) to 15 (extremely noisy). As the speckle noise variance increases from 0 to 0.36, performance decreases gradually due to low levels of noise distorting the object features quickly. After 0.36 variance, model performance keeps decreasing steadily as features are almost unrecognizable by the model even though objects are distinguishable by a human viewer.*

objects more adequately [46]. Speckle noise also seems to increase the contrast at the same time as adding noise compared to Gaussian noise as can be compared in Figures 3.8 and 3.9. This explains why the overall results are marginally higher than the results after applying Gaussian noise towards the ends of the ranges.

Adding differing amounts of speckle noise is an important method to use when testing the robustness of a deep learning model, because speckle noise introduces the effects of noise and contrast simultaneously. The acceptable amount is nonexistent since the performance drops when small amounts of speckle noise are added to the test images. As mentioned in Section 4.2.7, there are not many actions that can be taken towards fixing noisy imagery. The ones that do exist only slightly mitigate noise – adjusting color/luminance to simulate changing the exposure of the image.

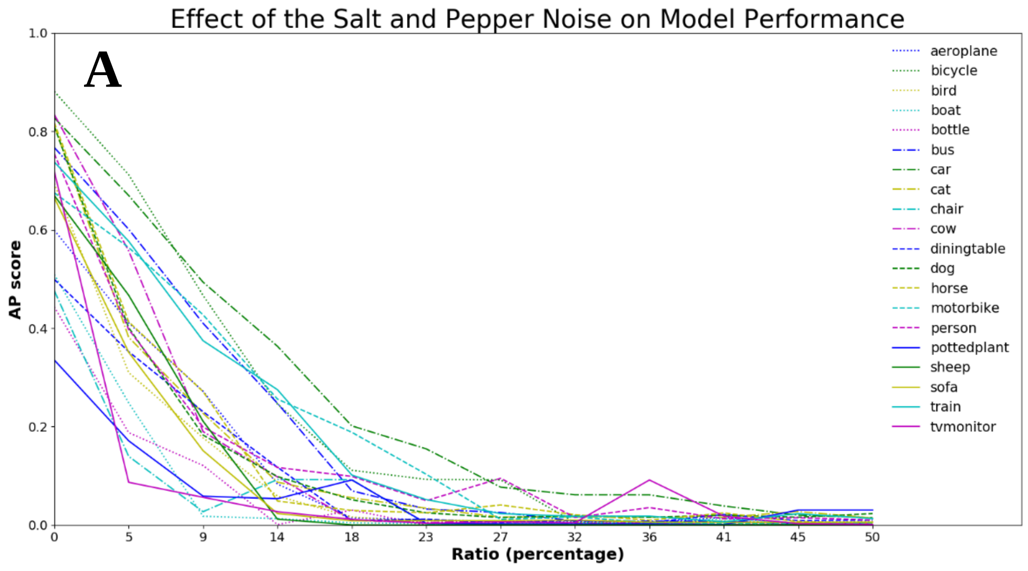
Briefly, speckle noise adds both noise and contrast to the images. This results in the model performing slightly better than it did on Gaussian noise, which is intuitively expected because adding contrast in small amounts makes the objects slightly more distinguishable while adding the noise sharply decreases performance.

4.2.9 Salt and Pepper Noise Results

Salt and pepper noise is the final method we applied to the images for testing the robustness of the model. The algorithm we use for applying this function can be found in [39] since it is not given by a simple equation. Similar to the previous two types of noise, this method was simple for us to choose a range to meet our criteria – visual differences are inherently easy to see by a human viewer and no midpoint needs to be included in the range. This allows us to keep the range continuous rather than having to split it into two sections. We create the range using the linear space starting at 0% and increasing to 50% using 4.17% increments. The points are 0%, 4.17%, 8.33%, 12.5%, 16.67%, 20.83%, 25%, 29.17%, 33.33%, 37.5%, 41.67%, 45.83%, and 50%. These values are rounded in Figure 4.10 for visual purposes.

The mAP scores for this function decrease sharply between 0% and 14% and slow down to a steady decrease for the reminder of the range, as shown in Figure 4.10b. The significant decrease in performance at the beginning is caused by the small amounts of salt and pepper noise distorting the object features within the image quickly. Once 14% of the noise is introduced to the images, the mAP score has markedly decreased from 0.677 at 0% noise to 0.116. Referencing back to Figure 3.10 in Chapter 3, the objects are still clearly recognizable by a human viewer at this level of noise. This indicates how quickly the noise alters the object features and how poorly the model handles this alteration. By the end of the range at 50% noise, the objects within the images are still mostly recognizable by humans, yet the model has a mAP score of 0.008, which means the model cannot correctly detect the majority of objects within the test imagery.

There are multiple unique results within Figure 4.10a, with the most notable being “car”, “bicycle”, “motorbike”, “bus”, “train”, and “tvmonitor”. The first five classes within those six all have a higher AP score from 5% noise to 18%. This is interesting because all of these are either vehicles or modes of transportation, and this phenomenon can be explained by the fact that all of these are



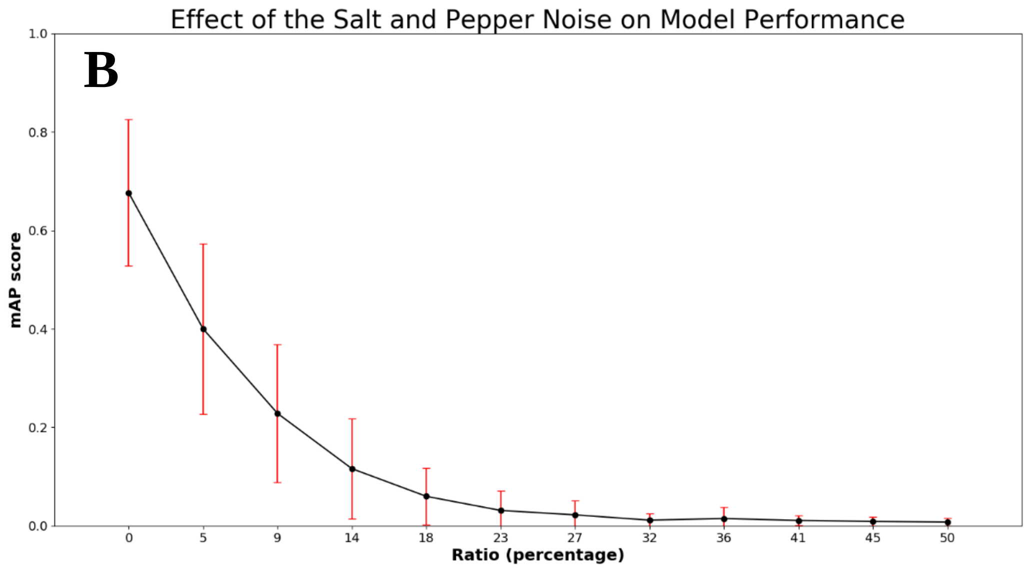


Figure 4.10 Effect of salt and pepper noise on model performance: (A) AP by VOC class and (B) mAP overall. *Model performance was measured for images applied with salt and pepper noise ranging from 0% (untransformed) to 50% (extremely noisy). As the salt and pepper noise increases from 0% to 14%, performance decreases sharply due to low levels of noise distorting the object features quickly. Above 14%, model performance keeps decreasing steadily as features are almost unrecognizable by the model even though objects are distinguishable by a human viewer.*

rigid objects in almost every case. This means that noise would not have as high of a detrimental effect on these object classes versus the others. The sixth class mentioned above – “tvmonitor” – stands out as well because it has the largest decrease in AP score when the salt and pepper noise is first introduced from 0% to 5%, dropping a total of 0.632 from 0.719 to 0.087. This is the largest amount a single object class has dropped within one time step in all of our experiments. There are two reasons for this: The VOC dataset only consists of 5.1% of the “tvmonitor” class in the training set [46], which is a small portion when checking for robustness; and as the salt and pepper noise is first introduced to the test imagery, the “tvmonitor” objects change from being a solid black or dark gray rectangle (when off) to having multiple pixels of differing colors from the noise, which would completely throw off the model. Outside of these six object classes, the other outliers exist towards the end of the range. These few classes that score higher than the others randomly, like “tv monitor” and “person” at 36% noise, can be mainly explained by the randomness within the GPU training on the VOC set [35].

Applying salt and pepper noise to the testing set of images is the last method we use to assess the model robustness. This method shows how inept the model is when this type of noise is introduced to the imagery. The performance quickly decreases as soon as the salt and pepper noise is included, unlike the previous noises, where the performance decreases much slower at first. The performance degrading this quickly indicates there is no range of acceptable salt and pepper noise levels. Similar to the Gaussian and speckle noises, the corrective actions to fix noisy imagery is almost non-existent. The actions that can slightly mitigate the noise are adjusting the color and luminance of the images, which would simulate changing the exposure. Overall, adding salt and pepper noise to the test imagery allows us to draw the following conclusions: the model is incapable of handling this type of noise adequately, and the model performs better on the other two previously mentioned types of noise than this one, but not by a large margin.

In short, as soon as the salt and pepper noise is added to the test images, the model performance degrades very quickly. This is to be expected since this type of noise is much more aggressive at changing the object features, as shown within the example image in Figure 3.10.

4.3 Sensitivity Analysis Discussion

Based on the overall results of our sensitivity analysis, we can place the nine degradation functions into three categories – high, medium, and low – based on the amount of effect or impact they had on model robustness. The high impact category contains brightness and contrast, shown in Figures 4.1 and 4.4 respectively. The medium effect group contains Gaussian blur, image size or pixelation, and all three types of noise – Gaussian, speckle, and salt and pepper – shown in Figures 4.5, 4.6, and 4.8-10. The low impact category contains saturation and sharpness, shown in Figures 4.3 and 4.7.

We chose these three categories based on how much variation there is within the graphs of each function. The functions which had a high impact on the model robustness are brightness and contrast. The graphs of both of these functions have almost the same derivative, which can be easily inferred from Figure 4.11. This follows what was mentioned in Chapter 3 about these being

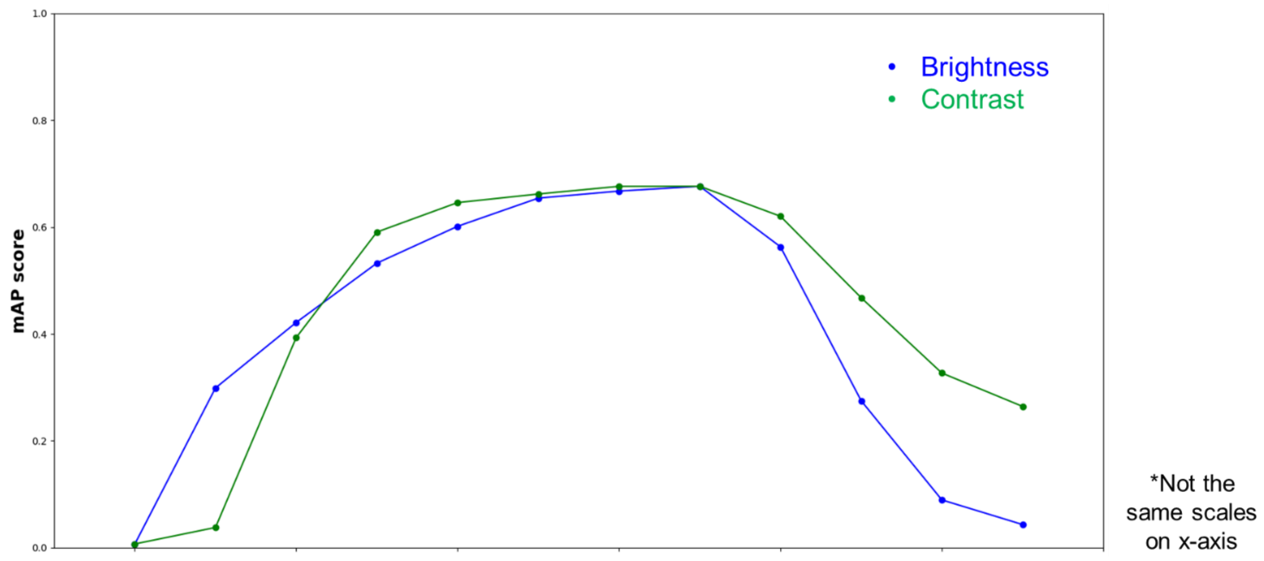


Figure 4.11 High impact degradation methods' results displayed in one graph. *Both the brightness and contrast results placed into one graph to allow a clearer depiction of why they are categorized as high impact. Both methods sharply decrease at the low and high ends of the range with a plateau in the middle.*

directly related to image intensity. The methods which had a medium amount of variation within their graphs are Gaussian blur, image size, and the three types of noise. The reason as to why they are categorized as having a medium effect on the model is because each of these functions only degrade the model performance rather than enhance it at some point. This is clearly depicted in Figure 4.12. The functions which had a low effect are saturation and sharpness, which are shown in Figure 4.13. These functions are also related to image intensity, but they do not follow the same amount of variation within their graphs as brightness and contrast – the other two components of image intensity. The most reasonable explanation for this occurrence is saturation and sharpness do not fundamentally have the same effect as brightness and contrast in an image. At high levels of saturation, the colors become purer, yet they are still distinguishable by the human eye. This is similar to high levels of sharpness where the edges of the objects become more distinct. At higher levels of brightness and contrast, the objects become vastly more undistinguishable.

Overall, these nine functions help determine which kinds of degradation are more important to consider when performing model inference to detect objects. The two most important functions are brightness and contrast, because these have the most impact on how well the model detects objects. The least important functions are saturation and sharpness since these techniques have negligible effects on the model.

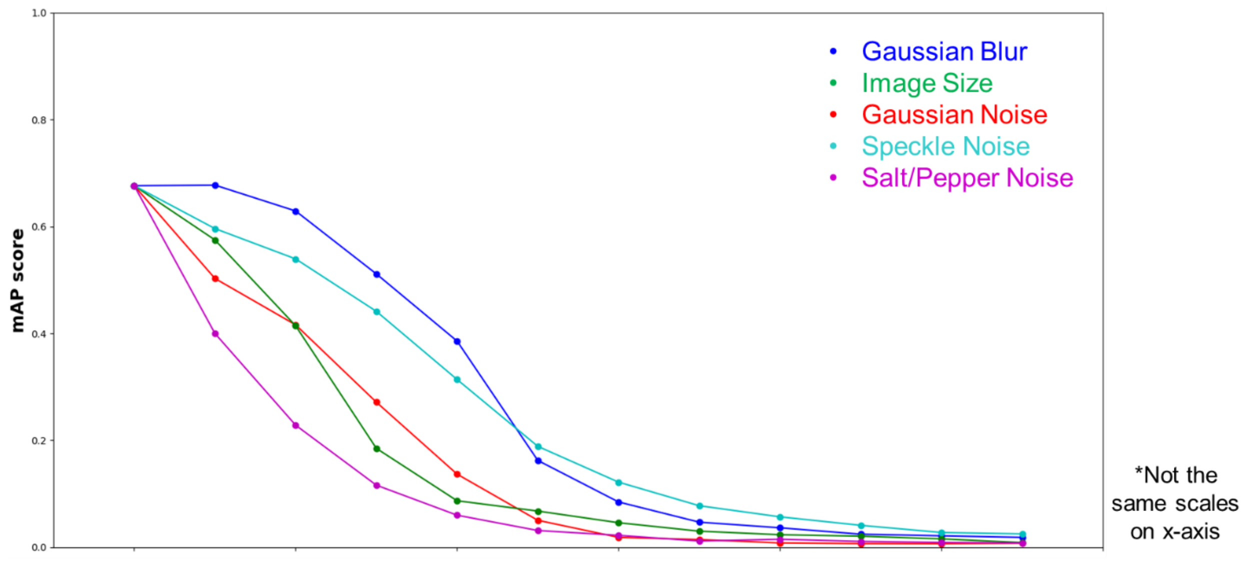


Figure 4.12 Medium impact degradation methods' results displayed in one graph. *All five of the methods' results placed into one graph to allow a clearer depiction of why they are categorized as medium impact. The Gaussian blur, image size, Gaussian noise, speckle noise, and salt and pepper noise results are all very similar to an exponential decay curve with salt and pepper noise being the most similar.*

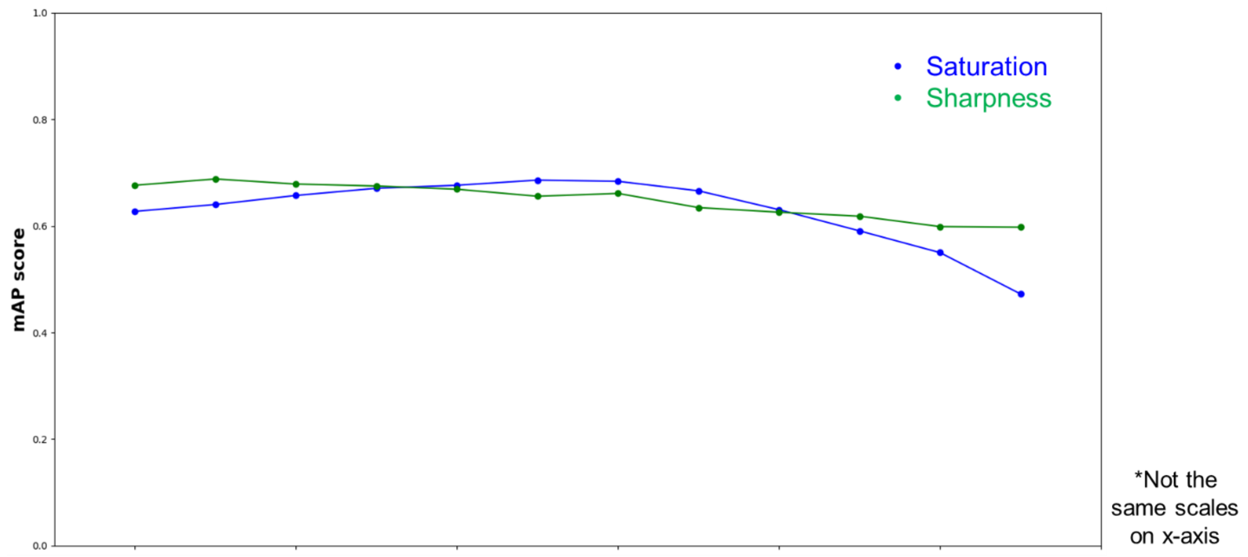


Figure 4.13 Low impact degradation methods' results displayed in one graph. *Both the saturation and sharpness results placed into one graph to allow a clearer depiction of why they were categorized as low impact. Both methods' results contain only a small variation through the entire range, which means they do not impact the model scores much at all.*

5 Conclusions and Recommendations

During the course of this research, we have shown how an object detection model performs with a custom dataset and how it performs when faced with degraded imagery. This chapter will discuss the actions we have taken and the results we have achieved throughout our work. After this, we will include further improvements for making our pipeline more robust in the future.

5.1 Key Points

In this thesis, we created two pipelines, as shown in Figure 1.1. The first is for training and testing an object detection deep learning model on a custom dataset. The second pipeline is similar to the first except we apply a degradation model to the testing images before we test the model, and we also use a publicly available dataset rather than our custom dataset in an attempt to provide results on more object classes. Both pipelines are implemented with the final result telling us how robust the model is. In this section, we will be giving an overview of both pipelines.

The first process pipeline starts with splitting our custom dataset – Aphylla – into training, validation, and testing sets. Once we get the three sets, we feed the training set into the object detection network to train it. We use the validation set to validate how well the network is trained and to determine if the network needs to be re-trained. We repeat this process until the network has adequately learned the object features, which results in the trained model. Then, we apply the trained model onto the testing set of images to determine how well the model performs on a set of images, which have not been seen by the model. This process outputs the detection results, which we then perform benchmarking on to determine the overall accuracy of our trained model. The final results we achieved are listed in Table 4-1, with the most notable result being we achieved a 98.5% mAP score using our custom dataset.

After training the network solely on our custom dataset, we decided to take it one step further and combine our dataset with the VOC2007 dataset to train the network on a combination of both. We repeat the same process mentioned above to obtain the mAP score of the model, which is 57.3% mAP as shown in Table 4-1. This result was surprising compared to the previous results. We determined the issue causing such a low mAP score, which is model overfitting, because the "dragonfly" class alone got a 96.8% AP score. The overfitting comes from the non-proportionate combination of the two datasets. There will be a recommendation in the Section 5.3 pertaining to the future work idea that could appropriately fix this. We did not have enough time to fix this issue once we determined the root cause of it.

The second pipeline we created follows a similar path as the first except we use the VOC2007 dataset for training the network and apply multiple different degradation models to the testing set of images to determine the overall robustness of the trained model on degraded imagery. We decided to use the VOC2007 dataset because it has a total of 20 object classes plus the background class, whereas our custom dataset consists of one object class – "dragonfly" – plus the background. This allowed us to obtain overall more results – 20 object classes versus one. We train the network using non-degraded imagery to obtain a trained model, similar to the first pipeline. Our main goal for this pipeline is to determine the robustness of this model when performing inference on degraded imagery. We chose nine different degradation models to apply to the testing set of images, which are brightness, contrast, saturation, Gaussian blur, image resizing, sharpness, Gaussian noise, speckle noise, and salt and pepper noise.

After choosing the types of degradation we were going to apply, we decided on using 12 different levels in each function based on the computational time it would take. The computational time was still far too long since we needed to apply nine different degradation functions with 12 different levels each onto the entire VOC2007 testing set of 4,952 images. So, to reduce the number of computations even further, we decided to reduce the number of images in the testing set. After doing some analysis and testing, we determined using 512 testing images produced a mAP score within 5% of the original baseline mAP score – 67.6% mAP using 512 images versus 71.1% using the original 4,952 images. This meant after applying all degradation functions with 12 levels each, we had a total of 55,296 images rather than having 534,816 images to perform inference on. We reduced the number of images down to almost 10% of the original amount. The computational time this many images take to perform inference on is about 10 hours.

Once we finished performing inference on all of the images, we obtained the detection results on the degraded imagery. With these detection results for each degradation type, we applied the benchmarking metrics to obtain the AP scores for each object class as well as the mAP scores. We graphed all of these results to visualize the impact each of these functions have on the model. We, then, sorted all of the degradation techniques into three categories: high impact, medium impact, and low impact. Brightness and contrast are the two most impactful functions on model robustness, having both a positive and negative impact on the mAP score over the 12 levels. The degradation techniques having a medium impact on the model are all three types of noise, image resizing, and Gaussian blur. These functions had a negative impact on the mAP score over the 12 different levels. The two functions having a low impact on model robustness are saturation and sharpness. These only varied the mAP scores very slightly over the entire range.

5.2 Contributions

The main contributions in this work are three-fold: (1) creating the framework/pipeline for determining model robustness, the whole of Figure 1.1; (2) the creation of the degradation model that is applied to the testing set of images before testing the model, the purple box in Figure 1.1; and (3) consistent guidelines and recommendations for mitigating the performance drops from the degradations. The first contribution will allow future extensions of this work to flow smoothly and produce robustness results with minimal effort. The second will allow any type of degradation to be applied onto the testing images for testing model robustness. The third will provide the user with guidelines about how to alleviate the performance drops caused by the degradation functions.

CONTRIBUTIONS:

1. Creation of the framework/pipeline
    a. Allows any dataset

    b. Allows any network
    c. Allows any model
2. Creation of the degradation model
    a. Nine degradation functions
    b. Allows any type of degradation
3. Consistent guidelines and recommendations for mitigating performance drops

The pipeline we created is the overarching contribution within this thesis. This includes the degradation model we created. This pipeline lays down the groundwork needed for testing model robustness with any base network and any object detection model available, whether that be in past, present, or future networks and models. This allows this thesis work to be used by anyone wanting to test for any type of model robustness on any kind of degradation techniques. In this thesis, we choose to demonstrate our pipeline using our binary class, custom dataset – Aphylla – as well as using the multi-class PASCAL VOC2007 dataset and using the VGG16 base network with the Faster RCNN object detection model. These are examples of the pipeline being used, even though any network and model can be used.

The degradation model we created sits in the pipeline between the testing set of images and the trained model and is our second major contribution. This model allows for any type of degradation to be applied to the testing images, and conversely, to the training and validation sets of images too. This will allow future works to be able to use our degradation model to test for many different types of model robustness beyond what we test in this research. This thesis applies 12 levels of nine different degradation types to the test set of images. We chose these nine types to demonstrate how our model integrates in the pipeline and show how simple it is to take one of the degradation techniques and change it to another.

The consistent guidelines and recommendations for mitigating performance drops we provide is our third major contribution. The three categories we classify each degradation type into are high, medium, and low impact. These categories are based on the fluctuations within the results. The high category resembles a Gaussian-like bell shaped curve that has a plateau at the top. This plateau covers a decent portion in both the brightness and contrast graphs, from 50% to 200%. Outside of this plateau, the fringes of the graph resemble exponential decays. An example of the guideline we provide based on these characteristics is at which point in the range would a pre-processing step need to be performed on the images before object recognition can be performed in order to minimize the performance decrease. For instance, in the second category, the results resemble mild to severe exponential decays throughout the entire range. In order to delay the model performance effects of these types of degradation, a pre-processing step, such as median filtering, needs to be applied to the test images. For the third class, the results show a small reduction in object detection performance, which stays mostly uniform throughout the range. This means that no action is needed.

5.3 Conclusions

In regard to training the model using our custom dataset, the mAP score turned out to be very high at 98.5%. After performing this test, we decided to combine our dataset with the VOC2007 dataset, which resulted in a much lower mAP score at 57.3%. The AP score for the "dragonfly" class alone was 96.8%. The conclusion we were able to draw from these results is both the models overfitted on the datasets. This was discovered too late for us to be able to fix this issue within this research. We have highlighted a way to fix this issue in the next section.

In the second section of the thesis, based on the overall results of our sensitivity analysis, we can place the nine degradation functions into three categories – high, medium, and low – based on the amount of effect or impact they had on model robustness. We chose these three categories based on how much variation there is within the graphs of each function. The high impact category contains brightness and contrast. The medium effect group contains Gaussian blur, image size or pixelation, and all three types of noise – Gaussian, speckle, and salt and pepper. The low impact category contains saturation and sharpness.

Overall, these nine functions help determine which kinds of degradation are more important to consider when performing model inference to detect objects. The two most important functions are brightness and contrast, because these have the most impact on how well the model detects objects. The least important functions are saturation and sharpness since these techniques have negligible effects on the model.

- High impact methods performed exactly as expected
    - Rising up from 0%
    - Plateauing in the middle
    - Dropping at the end
- Medium impact methods did not perform as expected
    - Resemble an exponential decay
    - Expected to perform well over the majority of the range, then steeply drop off at the end once human viewers are not able to recognize the objects
        - Human viewers are able to distinguish the objects throughout the entire range until the last couple steps
        - Deep learning model performs the exact opposite of human viewers in this category
- Low impact methods are mixed in meeting expectations
    - Sharpness did perform as expected
        - Objects do not get degraded in a similar fashion as the other methods
        - Edges are enhanced as the sharpness increases
    - Saturation did not perform as expected
        - Resembles a nearly linear line with a slight decline at the beginning and end
        - Expected to perform similarly to brightness and contrast since they are all related to image intensity

5.4 Future Work

While researching and developing this pipeline, we came up with many ideas for future improvements to our work. We will present them in this section, such as other types of object detection models, reducing model overfitting, minimizing model inference overhead, applying the degradation techniques simultaneously to the test images, and more degradation methods.

We designed our pipeline with the idea of being able to change the object detection we used with other object detection models rather than only being able to use Faster RCNN. We chose Faster RCNN to prove that our pipelines work (1) for training on a custom dataset and (2) for testing model robustness over degraded imagery. In the future, the other models to integrate into our pipeline would be YOLO [52], SSD [64], R-FCN [65], and maybe even Mask RCNN [30] even though it is an instance segmentation model. There are many other object detection models not mentioned here that could work also. Using different models would allow us to potentially get better detection results as more and more models come out and proceed to obtain higher and higher detection rates.

The next recommendation we had is balancing the image set sizes to reduce the chances of overfitting the model on the current dataset. This would allow the model to have better performance on a variety of different datasets rather than only the dataset the model was trained on. The model we trained using our custom dataset combined with the VOC2007 dataset overfitted to the point where it would detect dragonflies on almost all objects, including some parts of the background of the VOC2007 images. This idea of balancing the datasets would include splitting the custom dataset into more appropriately sized training, validation, and testing sets. VOC2007 uses a 25%, 25%, 50% split, whereas we used a 50%, 25%, 25% split. Another factor would also be the number of images within the dataset. The largest object class in the VOC2007 dataset is the "person" class with 2008 training and validation images [46]. Our custom dataset only contains the "dragonfly" class, and with how we split the dataset, there are 11,055 training and validation images. This skewed the model heavily towards the "dragonfly" class rather than the 20 classes in VOC. In the future, we would like to proportionately split the custom dataset to match the VOC2007 dataset in size and object class instances, which would take many more training attempts and many more tests to determine the optimal size to not overfit.

Computational overhead is a major obstacle in the field of deep learning, image processing, and computer vision. Computational overhead increases how long computations take and how many resources are consumed during the computation. The overhead in our project mostly comes into play when we perform model inference on the degraded test images. We have a total of 55,296 degraded images, which comes from our nine degradation functions, 12 iterations of each function, and 512 testing images. It takes about 3 minutes to run 512 degraded images through the model, which is one iteration of one degradation function. When we ran all 55,296 images through the model in one sequential run, it took about 10 hours to perform inference. This is almost double what it would have taken if we manually ran the model on each 512 image set. This overhead is from memory usage not be cleared properly after each run. In the future, one way to mitigate this would be to add a few seconds between each inference for Python to natively clear the memory. Other ways to mitigate this issue would need to be researched more extensively to find a solution.

The last two ideas relate specifically to the degradation model portion of our pipeline. The first recommendation would be to apply multiple different methods simultaneously to the images. In our work, we only applied the methods separately to the testing set of images to reduce the computational time performing model inference over all the images would take. Applying the methods in different combinations on the testing images would allow us to gather more results, which would give us a deeper understanding of model robustness. The second idea would be to implement multiple different degradation functions not used in this thesis. These would include bokeh blur, different methods of image compression, modifying the HSV color space in more ways, performing more variations of Gaussian blur and all types of noises, as well as any other degradation technique not mentioned here. This would provide more results and allow us to draw more conclusions about model robustness and how to overcome bad performance on degraded imagery.

List of References

[1] A. Maity, A. Pattanaik, S. Sagnika, and S. Pani, “A Comparative Study on Approaches to Speckle Noise Reduction in Images,” in *2015 International Conference on Computational Intelligence and Networks*, Odisha, India, 2015, pp. 148–155.

[2] R. Collobert and J. Weston, “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning,” in *Proceedings of the 25th International Conference on Machine Learning*, New York, NY, USA, 2008, pp. 160–167.

[3] X. Chen and A. Gupta, “An Implementation of Faster RCNN with Study for Region Sampling,” *arXiv:1702.02138 [cs]*, Feb. 2017.

[4] S. A. Clukey, “Architecture for Real-Time, Low-SWaP Embedded Vision Using FPGAs,” Dec. 2016.

[5] T. Ragland, “Automated Visual Monitoring of Machining Equipment,” Dec. 2011.

[6] A. Krizhevsky, V. Nair, and G. Hinton, “CIFAR-10 and CIFAR-100 datasets.” [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html. [Accessed: 09-Nov-2017].

[7] “CS231n Convolutional Neural Networks for Visual Recognition.” [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 12-Aug-2018].

[8] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[9] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” *arXiv:1311.2901 [cs]*, Nov. 2013.

[10] G. Hinton *et al.*, “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[11] C. Szegedy, A. Toshev, and D. Erhan, “Deep Neural Networks for Object Detection,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2553–2561.

[12] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning,” *arXiv:1712.06567 [cs]*, Dec. 2017.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015.

[14] S. Deivalakshmi, S. Sarath, and P. Palanisamy, “Detection and removal of Salt and Pepper noise in images by improved median filter,” in *2011 IEEE Recent Advances in Intelligent Computational Systems*, 2011, pp. 363–368.

[15] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[16] J. Yim and K.-A. Sohn, “Enhancing the Performance of Convolutional Neural Networks on Quality Degraded Datasets,” 2017, pp. 1–8.

[17] R. Girshick, “Fast R-CNN,” Apr. 2015.

[18] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *arXiv:1506.01497 [cs]*, Jun. 2015.

[19] E. David and I. Greental, “Genetic Algorithms for Evolving Deep Neural Networks,” *arXiv:1711.07655 [cs, stat]*, pp. 1451–1452, 2014.

[20] C. Szegedy *et al.*, “Going Deeper with Convolutions,” *arXiv:1409.4842 [cs]*, Sep. 2014.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Ha, “Gradient-Based Learning Applied to Document Recognition,” 1998.

[22] Z. Chen, W. Lin, S. Wang, L. Xu, and L. Li, “Image Quality Assessment Guided Deep Neural Networks Training,” *arXiv:1708.03880 [cs]*, Aug. 2017.

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

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[25] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *arXiv:1409.0575 [cs]*, Sep. 2014.

[26] "ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012)." [Online]. Available: http://www.image-net.org/challenges/LSVRC/2012/results.html. [Accessed: 17-Aug-2018].

[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[28] N. Barkmeyer, "Learning to recognize new objects using deep learning and contextual information," 05-Sep-2016. [Online]. Available: http://web.ics.ei.tum.de/~karinne/Pdfs/MasterarbeitICS-FINALVERSION-Niklas.pdf. [Accessed: 27-Aug-2018].

[29] A. Kloss, D. Kappler, H. P. A. Lensch, M. V. Butz, S. Schaal, and J. Bohg, "Learning where to search using visual attention," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea, 2016, pp. 5238–5245.

[30] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *arXiv:1703.06870 [cs]*, Mar. 2017.

[31] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[32] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, Oct. 2017.

[33] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," May 2014.

[34] H. A. Piwowar and W. W. Chapman, "Public sharing of research datasets: a pilot study of associations," *J Informetr*, vol. 4, no. 2, pp. 148–156, Apr. 2010.

[35] R. Luo, *pytorch-faster-rcnn*. 2017. [Online]. Available: https://github.com/ruotianluo/pytorch-faster-rcnn. [Accessed: 09-Nov-2017].

[36] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, Jan. 2016.

[37] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[38] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *arXiv:1311.2524 [cs]*, Nov. 2013.

[39] S. van der Walt *et al.*, "scikit-image: image processing in Python," *PeerJ*, vol. 2, p. e453, Jun. 2014.

[40] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective Search for Object Recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, Sep. 2013.

[41] S. Gai, B. Zhang, C. Yang, and L. Yu, "Speckle noise reduction in medical ultrasound image using monogenic wavelet and Laplace mixture distribution," *Digital Signal Processing*, vol. 72, pp. 192–207, Jan. 2018.

[42] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[43] J. Huang *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," Nov. 2016.

[44] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual Path Networks," *arXiv:1707.01629 [cs]*, Jul. 2017.

[45] R. Dekker, "The importance of having data-sets.," p. 5, 2006.

[46] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[47] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko, "Translating Videos to Natural Language Using Deep Recurrent Neural Networks," *arXiv:1412.4729 [cs]*, Dec. 2014.

[48] S. Dodge and L. Karam, "Understanding How Image Quality Affects Deep Neural Networks," *arXiv:1604.04004 [cs]*, Apr. 2016.

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

[50] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” 2017, pp. 6517–6525.

[51] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” *arXiv:1804.02767 [cs]*, Apr. 2018.

[52] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” *arXiv:1506.02640 [cs]*, Jun. 2015.

[53] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[54] M. Nielson, *Neural Networks and Deep Learning*. Determination Press, 2015.

[55] R. Gonzalez and R. Woods, *Digital Image Processing*, Second. Prentice Hall, 2002.

[56] A. Clark, “Pillow — Python Imaging Library,” *Pillow — Pillow (PIL Fork) 5.4.1 documentation*, 2018. [Online]. Available: https://pillow.readthedocs.io/en/stable/. [Accessed: 09-Nov-2017].

[57] “Amazon Mechanical Turk,” 2018. [Online]. Available: https://www.mturk.com/. [Accessed: 09-Nov-2017].

[58] J. Abbott, “Odonata Central: An online resource for the distribution and identification of Odonata,” 2019. [Online]. Available: https://www.odonatacentral.org/. [Accessed: 09-Nov-2017].

[59] W. Kuhn, “Zen of Dragons — Zooniverse.” [Online]. Available: https://www.zooniverse.org/projects/willkuhn/zen-of-dragons. [Accessed: 09-Nov-2017].

[60] “PyTorch.” [Online]. Available: https://www.pytorch.org. [Accessed: 09-Nov-2017].

[61] “Loss Functions.” [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. [Accessed: 09-Nov-2017].

[62] “rbgirshick/py-faster-rcnn - Issue #243,” *GitHub*. [Online]. Available: https://github.com/rbgirshick/py-faster-rcnn/issues/243. [Accessed: 09-Nov-2017].

[63] “Faster R-CNN (PyTorch implementation).” [Online]. Available: https://github.com/deboc/py-faster-rcnn. [Accessed: 09-Nov-2019].

[64] W. Liu et al., “SSD: Single Shot MultiBox Detector,” arXiv:1512.02325 [cs], vol. 9905, pp. 21–37, 2016.

[65] J. Dai, Y. Li, K. He, and J. Sun, “R-FCN: Object Detection via Region-based Fully Convolutional Networks,” arXiv:1605.06409 [cs], May 2016.

[66] W. R. Kuhn, “Three approaches to automating taxonomy, with emphasis on the Odonata (dragonflies and damselflies),” Rutgers University - Graduate School - Newark, 2016.

Vita

Matthew B. Seals was born in Morristown, Tennessee, where he spent all of his childhood. He graduated from Morristown West High School in 2011. He then attended the University of Tennessee in Knoxville and graduated with a Bachelor of Science degree in 2015. After graduating, Matthew started an internship at Oak Ridge National Laboratory.

After the internship, Matthew decided to return to University of Tennessee at Knoxville in pursuit of a Master of Science degree. He began working as a Teaching Assistant for the Computer Science 100 class as an on-going job throughout graduate school. He also returned to ORNL as an intern, which lasted until finishing graduate school. He completed graduate school with a Master of Science degree in Computer Engineering in the summer of 2019 while simultaneously obtaining four years of experience from ORNL.