Data Driven Style Transfer for Remote Sensing Applications

by

Evan A. Stump

Department of Electrical and Computer Engineering
Duke University

Date:_______________________

Approved:

___________________________
Leslie M. Collins, Supervisor

___________________________
Kyle Bradbury

___________________________
John Board

___________________________
Loren Nolte

___________________________
Stacy Tantum

Dissertation submitted in partial fulfillment of
the requirements for the degree of Doctor
of Philosophy in the Department of
Electrical and Computer Engineering in the Graduate School
of Duke University

2022
ABSTRACT

Data Driven Style Transfer for Remote Sensing Applications

by

Evan A. Stump

Department of Electrical and Computer Engineering
Duke University

Date: __________________________

Approved:

_____________________________
Leslie M. Collins, Supervisor

_____________________________
Kyle Bradbury

_____________________________
John Board

_____________________________
Loren Nolte

_____________________________
Stacy Tantum

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering in the Graduate School of Duke University

2022
Abstract

Recent recognition models for remote sensing data (e.g., infrared cameras) are based upon machine learning models such as deep neural networks (DNNs) and typically require large quantities of labeled training data. However, many applications in remote sensing suffer from limited quantities of training data. To address this problem, we explore style transfer methods to leverage preexisting large and diverse datasets in more data-abundant sensing modalities (e.g., color imagery) so that they can be used to train recognition models on data-scarce target tasks. We first explore the potential efficacy of style transfer in the context of Buried Threat Detection using ground penetrating radar data. Based upon this work we found that simple pre-processing of downward-looking GPR makes it suitable to train machine learning models that are effective at recognizing threats in hand-held GPR. We then explore cross modal style transfer (CMST) for color-to-infrared stylization. We evaluate six contemporary CMST methods on four publicly-available IR datasets, the first comparison of its kind. Our analysis reveals that existing data-driven methods are either too simplistic or introduce significant artifacts into the imagery. To overcome these limitations, we propose meta-learning style transfer (MLST), which learns a stylization by composing and tuning well-behaved analytic functions. We find that MLST leads to more complex stylizations without introducing significant image artifacts and achieves the best overall performance on our benchmark datasets.
# Contents

Abstract .................................................................................................................................iv

List of Tables ............................................................................................................................ viii

List of Figures ........................................................................................................................... ix

Acknowledgements .................................................................................................................. xii

1. Introduction .......................................................................................................................... 1

2. Background .......................................................................................................................... 8

2.1. Machine Learning ......................................................................................................... 8

2.1.1. Common Machine Learning Tasks ........................................................................... 10

2.1.2. Model Evaluation ..................................................................................................... 12

2.1.3. Regression ................................................................................................................ 13

2.1.4. Classification ............................................................................................................. 13

2.1.5. Detection .................................................................................................................. 16

2.1.6. Linear versus Nonlinear Models .............................................................................. 18

2.2. Deep Learning and Neural Networks ............................................................................ 20

2.2.1. Basic Network Components .................................................................................... 20

2.2.2. Training with Backpropagation .............................................................................. 25

2.2.3. Common Models ...................................................................................................... 26

2.3. Image Stylization .......................................................................................................... 29

2.3.1. CycleGAN ............................................................................................................... 33

2.3.2. CyCADA .................................................................................................................. 36

2.3.3. WCT2 ...................................................................................................................... 37
2.4. Automatic Augmentation ................................................................. 39
2.5. Summary ...................................................................................... 44

3. Buried Threat Detection .................................................................. 45
3.1. Introduction to BTD ................................................................. 45
3.2. Typical BTD Processing .......................................................... 47
3.3. Evaluating BTD Algorithms .................................................. 50
3.4. GPR Data .................................................................................... 51
3.4.1. GPR Data Features ............................................................ 54
3.4.2. Model Implementation ......................................................... 56
3.5. Experiments ................................................................................. 57
3.5.1. Data Supplementation With Target Data ......................... 62
3.5.2. Data Supplementation With Non-Target Data .................. 64
3.5.3. Data Supplementation With Target and Non-Target Data .... 67
3.6. Summary ...................................................................................... 68

4. Neural Style Transfer for Infrared Object Detection ......................... 70
4.1. Background ................................................................................. 70
4.2. Related Work ............................................................................. 74
4.3. Datasets ...................................................................................... 75
4.4. Model Implementations ............................................................ 77
4.5. Neural Style Transfer for Pretraining ...................................... 79
4.5.1. Experimental Design ............................................................ 79
4.5.2. Experimental Results ............................................................ 80
List of Tables

Table 3.5.1: Performance results when varying $N$, while $P = 0, 100$ for DL-GPR threat (T) alarms. The best-performing classifier is bolded. ................................................................. 64

Table 3.5.2: Performance results when fixing $N\% = 20$, while varying $P$ for DL-GPR threat (T) alarms. The best performing classifier is bolded. ................................................................. 64

Table 3.5.3: Results of varying, $N$, while $P = 0, 100$ for DL-GPR non-threat (NT) alarms. The best performer is bolded. ........................................................................................................ 66

Table 3.5.4: Results of fixing $N\% = 10$, while varying $P$ for DL-GPR non-threat (NT) alarms. The best performer is bolded ........................................................................................................ 66

Table 4.3.1: This table lists summary statistics of each benchmark dataset. ....................... 77

Table 4.5.1: This table shows the results of the experiment described in 4.4.1. ............... 81

Table 4.6.1: This table shows the result of our experiments with CMST methods described in 4.5.2, the winner in each category is bolded. ................................................................. 85

Table 5.3.1: This table shows the function dictionary available to MLST in our experiments................................................................................................................................. 100

Table 5.4.1: This table shows the result of our experiments with unsupervised CMST methods, the winner in each category is bolded. ................................................................. 102

Table 5.4.2: This table shows the result of our experiments with supervised CMST methods, the winner in each category is bolded. ................................................................. 103

Table 5.5.1: This table shows difference in performance using MLST for augmentation or for stylization........................................................................................................... 109
List of Figures

Figure 1.1: This figure shows an example of an RGB and an IR image of the same scene, in the left and right panels. .......................................................... 4

Figure 2.1.1: This figure shows a diagram of the training process for an ML algorithm. . 9

Figure 2.1.2: This figure shows an illustration of a ROC curve...................................................... 15

Figure 2.1.3: This figure shows an illustration of an PR curve. The area used to compute AP is shown in light blue................................................................. 16

Figure 2.1.4: This figure shows an illustration of IoU linking. A linked alarm (left) and unlinked (right) are shown for the typical threshold IoU=0.5........................................... 18

Figure 2.1.5: This figure illustrates the behavior of a linear classifier and a nonlinear classifier on a set of synthetic classification data......................................................... 19

Figure 2.2.1: This figure shows a diagram of a deep neural network that consists of an input layer, an output layer, and many hidden layers .................................................. 22

Figure 2.2.2: This figure shows a diagram of convolutional layers with a kernel size of three. Each neuron is only connected to the closest neurons in the previous layers. ......23

Figure 2.2.3: This figure shows an example of a max pooling layer................................................. 24

Figure 2.2.4: This figure shows an illustration of the calculation of error terms $\theta$ associated with each weight by backpropagation......................................................... 26

Figure 2.2.5: This figure shows an illustration of a CNN architecture for classification. . .27

Figure 2.3.1: This figure shows the architecture of the CycleGAN, for simplicity only the XY branch is shown. ................................................................. 34

Figure 2.3.2: This figure is an illustration of one direction of cycle consistency loss. The loss is the difference between the reconstructed and the original image. .................35

Figure 2.3.3: This figure shows an illustration of CyCADA. The architecture is the same as CycleGAN but with additional semantic consistency and task loss terms.................37

Figure 2.3.4: This figure provides an overview of the operation of the WCT algorithm. . .38

Figure 2.4.1: This figure illustrates the optimization strategy of AutoAugment............. 40
Figure 2.4.2: This figure provides an overview of Faster AutoAugment algorithm. The policy is updated by backpropagation based on the decision of the critic. 

Figure 2.4.3: This figure illustrates the objective of AutoAugment. The authors cast data augmentation as a process that fills missing points in the original distribution of data.

Figure 3.2.1: This figure shows a flowchart of a common BTD processing flow.

Figure 3.3.1: This figure provides an illustration of how detector alarms are a label for scoring by using a halo-based linking criteria.

Figure 3.4.1: This figure shows an example of a typical sweep pattern for a HH-GPR sensor. A B-scan, indicated in green is also shown.

Figure 3.4.2: This figure shows an illustration of the computation for our gradient visual feature.

Figure 3.5.1: This figure is a diagram of our experimental design. DL-GPR data is subsampled based on a selection criteria to create a supplementary dataset.

Figure 3.5.2: This figure provides an illustration of the procedure for creating the supplementary training dataset.

Figure 3.5.3: This figure shows the ROC curves for supplementation.

Figure 4.1.1: This figure shows an example of (a) an RGB image, (b) a real IR image and (c) a synthetic IR image produced by a style transfer model. Adapted from [22].

Figure 4.3.1: This figure shows an example image from each IR dataset. From left to right: FLIR, DSIAC, CAMEL, and KAIST.

Figure 4.5.1: This figure illustrates our experimental design. RGB imagery is stylized by a style transfer model to create synthetic IR images to pretrain a detection model.

Figure 4.6.1: This figure illustrates mixed batch training. RGB imagery is stylized by a style transfer model to create synthetic IR images, used to train a detection model.

Figure 4.6.2: This figure shows example imagery produced by each CMST method we evaluated.

Figure 4.6.3: Each row of images is a content image $I_c$, style image $I_s$, and output Image ($I$) triple.
Figure 4.6.4: This figure shows the results experiment in Figure 4.6.3 which have been expanded across more content images, and style images (bottom)...

Figure 4.6.5 This figure shows two particularly egregious example of CycleGAN altering the content of an image...

Figure 5.1: This figure shows an outline of our proposed stylization method, MLST......

Figure 5.2.1: This figure shows a diagram of our proposed MLST model. MLST learns a composition of functions for stylization using an adversarial critic ...

Figure 5.2.2: This figure shows a diagram of MLST training objective. MLST minimizes this distance in feature space between the real IR images and stylized RGB images......

Figure 5.5.1: This figure shows example imagery produced by each data driven CMST method. MLST introduces no artifacts and produces a complex nonlinear stylization.

Figure 5.5.2: This figure shows the final policy learned by MLST. In the limit \( k=1 \), MLST is equivalent to inverse grayscale, the best non data driven stylization.

Figure 5.5.3: This figure shows how MLST is trained for stylization (left) and how MLST is trained for augmentation (right).

Figure 5.5.4: This figure shows the differences in learned policies of MLST when used for stylization (blue), augmentation (orange), and unsupervised MLST.

Figure 5.5.5: This figure shows the UMAP manifolds of features from a YOLO model on Kaist for real IR, RGB, Augmented IR imagery, and MLST stylized imagery.

Figure A.1: This figure shows the UMAP manifolds of features extracted from a trained YOLO model on FLIR.
Acknowledgements

While working toward a PhD, I wanted to prove to myself and to others that I was worth all the hard work that my mentors put into me. For a long time, I didn’t think I was. During my tenure at Duke there were many times where I thought it was foolish to keep trying; but with the help of my advisors, my friends, my family, and others, I’ve come to learn that the last word had not been said. Defeat is not final, nor will hope disappear. For hope does not depend on the strength of one, but the bond between all.

Therefore, it is only right to acknowledge those who supported me in ultimately realizing this work. Starting with my colleagues in the Applied Machine Learning Lab: Kevin Chu, Bohao Huang, Emma LaPorte, Janet Chen, Lidia Shahidi, Francisco Reveriano, Joel Bjornstad, Kaleb Kassaw, Francesco Luzi, Gregory Spell, and Simiao Ren. For without their knowledge, and friendship I would not have the heart to have finished the program. I would also like to acknowledge the staff at CAPS and their associates, particularly Rebecca Hurst and David Oh.

To my advisors Jordan Malof and Leslie Collins, I could never adequately express the depth of my gratitude for the opportunity to learn from you both. When the ship had sailed and all seemed lost, you showed me that there was indeed hope. I find it in spontaneous late-night meetings turned to mutually therapeutic rants, the celebration of holidays; or in larger gestures such as giving aid in time of need, keeping promises, and offering guidance in an otherwise impossible task. Thank you, most sincerely.
1. Introduction

The fundamental goal of this work is to develop more accurate automatic target recognition (ATR) systems for remote sensing problems. Remote sensing is most broadly defined as the task of detecting and monitoring physical information from a distance. It is used whenever physical information would be directly inaccessible or unsafe to collect. ATR systems have been a subject of research for a large variety of remote sensing systems, and specific target objects. Some notable examples include buried threat detection (BTD) [1], and object detection in infrared (IR) imagery[2] [3].

Tremendous progress has been made in the field of remote sensing, often by developing highly tailored statistical or machine learning models that leverage the unique phenomenology and statistical properties associated with the sensing problem [3]. This need for problem specific solutions has led to a wide variety of different ATR algorithms, with widely varying processing strategies. In recent years however, deep learning methods – namely deep neural networks (DNNs) – have led to breakthrough performance across many different remote sensing tasks. Although it is still common to tailor solutions to the problem, many fields now rely primarily upon DNNs as their machine learning model of choice [4].

One key limitation of DNNs in the context of remote sensing, however, is their need for large, labeled training datasets to effectively learn the necessary model [5]. Addressing the demand for more data is the primary focus of this dissertation.
Unfortunately, it is not always a simple task to acquire more data. It can be difficult, time consuming, and prohibitively expensive to collect and annotate a sufficiently large dataset that is needed to train a deep model. Data for remote sensing tasks such as BTD and infrared (IR) object detection are much less ubiquitous than data for popular domains for machine learning applications involving natural imagery/video [6], automatic speech recognition [7], or other natural language processing tasks [7].

There are many general strategies to solve this problem: making more data-efficient models (e.g., ResNet [8]), making greater use of existing data (e.g., augmentation [9], self-supervised learning [10]), faster data collecting (e.g., synthetic data [11] [12], crowdsourcing data [13]), and transfer learning [14]. In this work, we focus on transfer learning. Simple forms of transfer learning are already widely used, and highly successful [14] [15] [16]. However, these techniques are relatively simple and do not take full advantage of knowledge gained on related tasks [17] [16] [18]. Therefore, in this dissertation more advanced and novel transfer learning approaches are considered.

Transfer learning is a machine learning technique which is focused on storing knowledge learned while solving one ‘source’ problem and applying it to a different, yet related, ‘target’ problem [14]. Transfer learning is most commonly implemented in the form of pretrained networks [14]. Pretrained networks are DNNs trained on a large dataset, most commonly ImageNet (a very large and diverse RGB dataset) [19]. The
model weights learned from ImageNet are saved, and then used as the initialization for training DNNs for a more specific task. One example of related source and target tasks would be RGB object detection and IR object detection respectively, as described below. Qualitatively, an RGB and an IR image of the same scene are related and the same types of semantic objects (e.g., the cyclist, the cars, and the lamp posts) in an image are similar, as is illustrated in Figure 1.1. The spatial statistics of the same types associated with semantic objects are similar across the different sensor modalities [20] [21]. We hypothesize then, that the spatial statistics learned from training using the RGB data will generalize to the problem of detecting objects in IR imagery. However, we hypothesize that the spectral statistics are less similar (compared to the spatial statistics) across the two sensing modalities. Because of this difference, we hypothesize that the standard transfer learning strategies are not ideal for this task. By transforming RGB data to more closely resemble IR data, an individual RGB datum becomes a more representative, and therefore a more valuable training example for IR detection algorithms. This transformation, also known as stylization, makes transfer learning more effective as a strategy for training DNNs [14]. In this work, we will present how this transformation may be learned directly from data.
Figure 1.1: This figure shows an example of an RGB and an IR image of the same scene, in the left and right panels. Qualitatively, the spatial features associated with the semantic objects are similar across domains. Adapted from [22].

In this thesis we address the demand for more training data in the context of remote sensing tasks, specifically BTD and infrared object detection. Both applications have a relatively limited supply of labeled training data. Our goal is to maximize the performance of detection algorithms for these target tasks. The approach that we adopted was to leverage large annotated datasets from related tasks and stylize them in such a way that we create ‘target-like’ data which more closely resembles data from our target tasks and thus improves our detection performance. From there, we use the stylized target-like data in conjunction with our existing real target data to train algorithms for the target detection task. This dissertation presents several contributions to the science of BTD and ATR systems with this shared goal. With these research strategies, the following contributions are made in chapters 3, 4, and 5:
• Leveraging large quantities of labeled downward-looking GPR data to improve landmine ATR algorithms for hand-held GPR. Based upon this work we found that simple pre-processing of downward-looking GPR makes it suitable to train machine learning models that are effective at recognizing threats in hand-held GPR as well. This work leverages many decades of data collection performed on downward-looking GPR to mitigate data collection for hand-held systems and was published in [23], [24].

• Leveraging very large quantities of labeled RGB datasets to train DNN-based infrared ATR models. In this work we found that stylizing RGB images to make them more closely resemble IR images makes this stylized data suitable for training deep learning models that are effective at recognizing objects in IR imagery. This approach leverages large, diverse RGB datasets to mitigate the need for IR data collection.

  o We conducted controlled experiments measuring algorithm performance across multiple benchmark IR detection datasets. Our experiments show that using stylized RGB imagery is beneficial to algorithm performance when used either in the pretraining or fine-tuning of models.

  o We present an objective comparison of several state-of-the-art stylization models using a performance on a downstream ATR task as an evaluation metric.
- Using meta-machine learning to stylize RGB datasets for training DNN-based infrared ATR models. Based upon this work we found that stylizing RGB images makes it suitable to train deep models that are effective at recognizing objects in IR imagery. Stylizing data in this way yields better performance on ATR tasks than state of the art stylization algorithms.

  o We developed a novel ‘IR realistic’ stylization to create IR versions of preexisting RGB datasets, like COCO [13]. This method of stylization offers superior performance on downstream detection tasks compared with state-of-the-art stylization algorithms.

  o We conducted controlled experiments characterizing different stylization functions learned on multiple different benchmark IR detection datasets.

The organization of this dissertation is as follows: chapter 2 explores some of the preexisting ideas and methods needed to develop the algorithms and methods proposed in this work. This includes a review of existing detection algorithms, definitions of scoring metrics, classification models, stylization models, and automatic augmentation models. Chapter 3 presents algorithms that were developed for hand-held GPR systems, as well as their performance results on real field collected sensor data. The results that will be presented show that using vehicle based GPR data in addition to hand-held GPR data results in improved performance on the hand-held GPR test dataset. Chapter 4 presents an algorithm that was developed for object detection on infrared imagery and
the performance on several benchmark datasets, an objective comparison that comprises several of state-of-the-art stylization models using a downstream detection task. The results that will be presented show substantial improvement for models trained with data augmented by the new stylization algorithm that was proposed across all benchmark datasets. Chapter 5 presents a novel stylization algorithm using learned data augmentation. The results that will be presented show improvement for models trained with data augmented by our algorithm perform favorably compared to state-of-the-art stylization algorithms across all benchmark datasets. Finally, chapter 6 discusses the conclusions and potential future work.
2. Background

This chapter provides a brief review of the essential concepts required to develop the algorithms presented in subsequent chapters. In section 2.1 a review of relevant machine learning tasks, evaluation methods, and concepts is provided. In section 2.2, a review of deep learning concepts is provided, as well as a review of some of the models which are used in chapters 4 and 5 of this work. Section 2.3 presents an overview of neural style transfer, and the particular models we utilize in chapter 4. In section 2.4 we discuss concepts of meta-learning and models related to automatic augmentation, which will be employed in chapter 5. In section 2.5 a summary of the chapter is provided.

2.1. Machine Learning

In our work we develop machine learning algorithms for remote sensing tasks. Specifically, we are developing algorithms for object detection in imagery. Machine Learning (ML) algorithms build a statistical model based on example data, known as ‘training data’ in order to generalize and make predictions on ‘testing data’. A ML algorithm generally has three fundamental features: a decision process, an error function, and a model optimization process. In the first step of training, a model will make some prediction based on input data. In this work all the ground truth labels for our data are known in advance. An error function can then make a comparison between this prediction and the ‘ground truth’ of the data to assess the accuracy of the model. This is known as supervised learning [5]. If the model can make better predictions for
data in the training set, the parameters, or ‘weights’, of the model are adjusted according to the model’s optimization process to reduce the error function. During training the algorithm will repeat the evaluation and optimization process several times until some stopping criterion has been met. This stopping criterion is typically convergence on an ideal solution that minimizes the error function for the training dataset [25]. A diagram of the training process is shown in Figure 2.1.1. However, there are some models that have a closed-form solution for estimating an ideal parameter (or set of parameters) which minimizes the error function [25]. An example of such a model is linear least squares [25].

![Figure 2.1.1: This figure shows a diagram of the training process for an ML algorithm. Given a training dataset, a ML algorithm will make predictions based on the model’s learned parameters. The predictions of the model are then evaluated according to an error function. If the stopping criteria is not met, the parameters of the model are updated according to the model’s optimization function to make better](image-url)
predictions on the training dataset. Once the stopping criteria has been met the final model can be used to make predictions on the testing dataset.

It is important to note that it is a fundamental assumption of ML that a model’s training data is representative of the testing data and that both datasets have similar underlying statistical properties. In general, ML algorithms do not generalize well to new data if the statistics of the test data are sufficiently different from those of the training data of the model [25].

2.1.1. Common Machine Learning Tasks

Many applications of ML, including those for remote sensing, are intended to solve variations of a few fundamental tasks. These fundamental tasks are reviewed in this section. Though their exact applications may differ, models designed for each task share many elements including their training objectives and functional form [5].

Regression is the task of estimating the relationships between an outcome variable and one or more input variables. A simple case of a regression problem is linear regression where a model estimates a function that maps an input to the real line (or in higher dimension problems, a hyperplane) [25]. Given an example dataset with a matrix of input variables $X$, and a vector of output variables $y$ the model assumes a linear relationship between $X$ and $y$. The model takes the form:

$$y = \beta X$$
$X$ is a matrix whose $ij$ element is the $i^{th}$ observation of the $j^{th}$ input variable, $y$ is a vector whose $i^{th}$ element represents the $i^{th}$ observation of the output variable, and $\beta$ is a vector of learned parameters. Regression models are commonly trained by ordinary least squares (OLS) [25]. OLS estimates model weights by minimizing the sum of squared differences between the prediction of the model and the known output variable. The training objective is of the form:

$$L = \sum_{i=1}^{n} (y - y^*)^2$$

where $y^*$ is the predicted output of the model and $L$ is the prediction error (or ‘loss’).

**Classification** is the task of assigning a categorical label or ‘class’ to input data. A classification algorithm must learn a discriminative function to predict a discrete class label $C_{ik}$ from an input data vector $x_i$ which is a row from $X$, $C_{ik}$ is a vector whose $i^{th}$ element represents the $i^{th}$ output class $C_k$, assigned to $x_i$, where $k$ is one of $K$ discrete classes such that $k = 1, ..., K$. These class labels correspond to semantic content, e.g. ‘dog’, ‘car’, or ‘person’. Mathematically, classification algorithms attempt to model the conditional probability distribution $p(C_k|x)$ and optimally infer $C_k$ from this distribution.

**Detection** is the task of classifying and localizing objects in an image. A detection algorithm draw a bounding box around the visible spatial extent of the detected objects. A bounding box is characterized by a four element vector: $[x, y, w, h]$ where $x, y$
represent the $x, y$ position of a box in the image (in pixel coordinates) and $w, h$ represent the width and height of the box in pixels. During training, a detection model learns to minimizes the difference between predicted boxes and ground truth boxes by minimizing a regression loss while also minimizing a classification loss. The detection algorithm must then predict a categorical class label $C_k$ associated with the object in the predicted box. This process is the same as the previously described classification task.

Other tasks associated with machine learning can be thought of as special applications of the previously mentioned three [25]. Two such tasks of interest to this work are style transfer [26], and automatic augmentation [27]. These tasks will be reviewed in sections 2.3 and 2.4 respectively.

2.1.2. Model Evaluation

To show the effectiveness of any proposed algorithms, we need a way to quantify and evaluate their performance. There are many ways to compute summary statistics which measure the general effectiveness of an algorithm. This is a two-step process. First predictions of the model are assigned a label either directly from the model or by a ‘linking’ function in detection problems. Then predictions are ‘scored’ where summary statistics are then computed. These steps are reviewed later in this section. For classification and detection problems, the summary statistics most often used to characterize models are ROC curves [28] and PR curves [4].
2.1.3. Regression

There are many summary statistics which measure the goodness of fit of a regression model [25]. Mean square error is an absolute measure of goodness of fit [25]. It is defined as:

\[
MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - y_i^*)^2
\]  

(2)

where \( y^* \) is the predicted output of the model and \( N \) is the total number of predictions.

2.1.4. Classification

For a given input a classification model outputs a categorical class label \( C_k \) and a confidence statistic which may be normalized to be between [0,1]. The predicted class may be correct or incorrect compared to the known ground truth of the data. The posterior probability of \( p(C_k|x) \) is given as:

\[
p(C_k|x) = \sigma(y_k^*) = \frac{\exp(y_k^*)}{\sum_j \exp(y_j^*)}
\]  

(3)

where \( y_j^* \) represents the output predictions of a model for class \( j \). \( \sigma(y_k^*) \) is known as the softmax function [25]. It is the multiclass generalization of the logistic sigmoid and represents a version of the \( \max() \) function that is continuously differentiable [25]. Cross entropy loss (CE), which is a commonly used loss criteria in classification tasks is defined as:

\[
CE(y^*) = - \sum_{k=1}^{K} y_k \log( \sigma(y_k^*) )
\]  

(4)
where $y$ represents the ground truth class and $y^*$ represents the output predictions of a model.

**Receiver Operating Characteristic (ROC) curves** quantify how well a model performed as a function of all possible confidence thresholds [28]. ROC curves are a common metric for evaluating detection and classification algorithms [28].

An ROC curve summarizes the performance of an algorithm on a binary classification problem for all classification thresholds. The curve plots two parameters: the probability of detection, and the probability of false alarm. The probability of detection, or true positive rate, is defined as $P_D = \frac{TP}{TP+FN}$. The probability of false alarm, or false positive rate, is defined as $P_{FA} = \frac{FP}{FP+FN}$. Where $TP$ is the number of true positives, $FN$ is the number of false negatives, and $FP$ is the number of false positives. For real-world problems, it is common to normalize the probability of false alarm by a metric such as false alarms per square meter. This is known as the false alarm rate, or FAR. $P_D$ and $P_{FA}$ are bounded on the interval [0,1]. A score of 1 represents perfect classification at a given threshold. FAR is bounded on the interval [0, $\infty$). A score of 0 represents no false alarms at a given threshold.

Sometimes an ROC is summarized with a single number an area under the curve (AUC) score. It is defined as the integral of the ROC with respect to $P_{FA}$ (or FAR). The AUC is a measure of model performance across all classification thresholds and thus provides a robust metric that can be used to compare how the performance of several
algorithms are ranked. Another common performance metric is a partial measurement of the AUC up to a specified FAR, sometimes known as pAUC. The pAUC is normalized by the maximum possible area between \( P_{FA0} \) and \( P_{FA1} \) so that pAUC is bounded by the interval [0,1]. These definitions are illustrated in Figure 2.1.2.

![ROC curve with highlighted AUC and pAUC areas](image)

**Figure 2.1.2:** This figure shows an illustration of a ROC curve. The area used to compute the AUC is shown in light blue and the area used for the pAUC measure is shown in darker blue, between the values of \( P_{FA0} \) and \( P_{FA1} \). The ROC curve for chance, or random, detection is also shown along the diagonal.

**Precision Recall Curves** (PR) curves are another common metric used for evaluation of algorithm performance. A PR curve plots two parameters: precision and recall. Precision is the proportion of correct predictions; it is defined as \( P = \frac{TP}{TP + FP} \).

Recall represents the proportion of objects found; it is defined as \( R = \frac{TP}{TP + FN} \). Both metrics are bounded between [0,1]. A high precision is indicative of good classification accuracy, and a high recall means that most objects were found. By convention, PR curves start at zero recall and a precision of one. As with ROCs, sometimes it is
convenient to summarize a PR curve with a single number. There are two common metrics: mean average precision $mAP@0.5$ and $F1$ score [4]. $mAP@0.5$ is the mean of the average precision (AP) across all classes at the IoU threshold 0.5. An example PR curve is shown in Figure 2.1.3. $F1$ score is the geometric mean of precision and recall, defined as: $F_1 = 2 \cdot \frac{(P \cdot R)}{(P+R)}$.

Figure 2.1.3: This figure shows an illustration of an PR curve. The area used to compute AP is shown in light blue.

### 2.1.5. Detection

The output of a detection algorithm is a vector of decision statistics and proposed object locations represented by a bounding box. Because a prediction rarely matches a label exactly each prediction of the model is ‘linked’ by a linking function. A linking function labels whether a prediction corresponds to a true object, or a false alarm based on some geometric criteria. By knowing the ground truth in advance and the prediction locations, each alarm location can be linked to being a true positive $TP$, a false positive $FP$, and a true negative $TN$. A linking function determines the link score between a prediction and a ground truth. A domain-specific linking function is used to determine if a predicted object is a true object, a false alarm, or a true negative.
There are two metrics in the linking processes: geometric proximity to ground truth, and classification threshold. If a detector’s decision statistic is low enough such that it falls beneath a classification threshold (which is a hyperparameter which sets the sensitivity of the detector) then that decision statistic is discarded.

Regarding proximity to ground truth, if a prediction is too far away from a ground truth annotation, it is considered a false alarm. This metric can be as simple as requiring a minimum distance (in pixels) to an annotation. Another common metric for this is Intersection over Union (IOU) scoring [29]. IoU is measured by the area of the intersection of a prediction box and a ground truth box, divided by the area of the union of those same boxes. An example IoU calculation is shown in Figure 2.1.4 Figure 2.1.5. IoU is bounded over the interval [0,1] with a higher score representing a more accurate prediction. Note that because IoU scoring criteria is a hyperparameter, the choice of threshold value can be more or less permissive as to what model predictions are considered hits. It is a common practice in the literature that the threshold to declare a correct decision is set to 0.5 IoU [29].
Once linking is complete, the predictions of a detection model have been mapped to a set of classification labels and confidence scores which can be evaluated with ROCs or PR curves.

If multiple predictions occur in close proximity to one another, it is common to only link the one with the maximum decision statistic and ignore the others such that they are not counted negatively. This practice is known as non max suppression. In multiclass detection problems, if a prediction occurs close enough to be linked to a ground truth annotation but is classified incorrectly it will not be counted as a correct detection in scoring.

**2.1.6. Linear versus Nonlinear Models**

Linear models are simple and computationally inexpensive, but they may be unable to map data vectors to their class labels if the true functional relationship is complex (i.e., nonlinear). In this case, nonlinear models are often better suited to the
decision-making task. Nonlinear models can learn a more complex representation of data. An example of the decision boundaries of a linear and nonlinear classifier are shown in Figure 2.1.5. However, nonlinear models are also prone to erroneously fitting to residual variation (noise) in data. This is known as overfitting and can be extremely detrimental to the performance of an algorithm in practice [25].

Figure 2.1.5: This figure illustrates the behavior of a linear classifier and a nonlinear classifier on a set of synthetic classification data. Each red dot corresponds to a point from Class 0, while each blue dot corresponds to a point from Class 1. A Linear Discriminant Analysis (LDA) would yield a solution in the form of a decision boundary that bisects the feature space as shown in the figure (dashed black). A support vector machine can create a more complicated decision boundary, in this case a circle (solid black line). Although the SVM can provide more complex solutions, this can also lead to overfitting of the data. Therefore, each of the two methods will work well under certain conditions of the data.
2.2. Deep Learning and Neural Networks

In the work presented in chapters 4 and 5 we use deep learning models to detect objects in IR camera data. Deep learning is a specific class of machine learning algorithms which most commonly focuses on Deep Neural Networks (DNNs). DNNs are the current state of the art for object detection algorithms [30]. Every model on the leaderboard of COCO, the largest public benchmark object detection dataset, is a DNN of some kind [31].

Though DNNs vary widely in architecture and function, they all share many common elements [14]. These building blocks common to deep object detection algorithms are reviewed in this section.

2.2.1. Basic Network Components

A DNN is a series of functional transforms made up of a hierarchical composition of basic nodes, termed neurons, that are organized into layers. Each neuron can be thought of as its own regression model composed of input data, weights, and an output [25]. A neuron has the form:

\[ a_m = \sum_{i=0}^{D} w_{mi}x_i \]  

(5)

where \( x_i \) and \( w_{ij} \) are the \( i^{th} \) inputs and weights to the \( m^{th} \) neuron, respectively, in the \( l^{th} \) network layer. The output of a neuron is known as an activation. Each activation is transformed by a differentiable nonlinear activation function, \( h \), to give a final output:
\[ z_m = h(a_m) \]  \hspace{1cm} (6)

Multiple neurons are connected in sequence to make a DNN. An example of a DNN is illustrated in Figure 2.2.1. As data moves deeper through the network, each neuron represents an increasingly complex and nonlinear function. The function approximated by a neuron in the second layer of a neural network is:

\[ z_{n+1}^l = h\left(\sum_{m=1}^{M} w_{mn}^{l+1} h\left(\sum_{i=1}^{D} w_{mi}^l x_i\right)\right) \]  \hspace{1cm} (7)

where \(z_{n+1}^l\) represents the \(n^{th}\) hidden variable in the \(l + 1\) layer of the network.

Ultimately, DNNs represent their input data as a learned, weighted, combination of nonlinear functions. In doing so, by the universal approximation theorem, a DNN of sufficient size can (theoretically) learn to approximate any function [32].
Figure 2.2.1: This figure shows a diagram of a deep neural network that consists of an input layer, an output layer, and many hidden layers. Each layer is composed of multiple neurons which are connected to other neurons in subsequent layers. The flow of information through the network is indicated by a green arrow.

As mentioned, neurons are organized into layers within a DNN. The exact configuration of connections between neurons in a layer allows for the creation of multiple types of layers, each with a specific function [5]. Object detection models are typically made of three types of layers: fully connected layers, convolutional layers, and pooling layers.

In a fully connected layer, every neuron is connected to every other neuron in the previous layer. When dealing with images as inputs, it is often impractical to connect every neuron to every neuron in the previous layer because fully connected layers do not account for any spatial structure of the input [5]. Convolutional layers enforce local connectivity between neurons in adjacent layers by only connecting neurons within a
specified ‘kernel size’. For example, in a convolutional layer with a kernel size of three, each neuron is connected to the three closest neurons from the previous layer.

Convolutional layers can be thought of as acting like convolution with a learnable filter in images (hence the name) [5] [33]. An example of a convolutional layer is shown in Figure 2.2.2:

![Diagram of convolutional layers](image)

**Figure 2.2.2:** This figure shows a diagram of convolutional layers with a kernel size of three. Each neuron is only connected to the three closest neurons in the previous layers.

DNNs composed primarily of convolutional layers are known as Convolutional Neural Networks (CNNs). The input to a convolutional layer is an $m \times m \times a$ image, where $m$ is smaller than the spatial extent of the image. The convolutional layer will have $k$ filters of size $n \times n \times b$ such that $n < m$. The output of the convolutional layer is $k$ convolutional feature maps of size $m - n + 1$. A nonlinear activation function is then applied to the output of a convolutional layer. The shallowest convolutional layers in the network are
responsible for capturing low-level features like edges, textures, or colors. Deeper layers capture high-level features which are more specific to the problem.

Pooling layers downsample their input features [33]. The pooling operation is specified instead of learned. The most common pooling function is max pooling [5]. Max pooling layers divide the input data into several patches and sample the maximum value of the data in each patch. The output features of a pooling layer represent a summarized version of the input features. Using pooling layers in a network architecture has several advantages. Pooling layers allow CNNs to be invariant to local spatial translations [33]. Additionally, pooling layers reduce the dimensionality of the data which reduces the overall number of parameters in the model. This allows for faster computation time and reduces the memory requirement of the model. An example of a max pooling layer is shown in Figure 2.2.3.

![Figure 2.2.3](image)

**Figure 2.2.3:** This figure shows an example of a max pooling layer. Input features to this layer are divided into patches; the patches are shown in orange, green, blue, and gray. The maximum value is sampled from each patch. The resultant output features represent a summarized version of the input features.
2.2.2. Training with Backpropagation

Neural networks are trained by gradient descent optimization via backward propagation of errors, or ‘backprop’ [34]. Gradient descent chooses the weight update to be a small step in the negative gradient of the error function, or ‘loss’ [34].

\[ w^{\tau+1} = w^{\tau} - \eta \nabla E(w^{\tau}) \]

where \( E \) represents the error of the network with weights \( w \) at iteration \( \tau \) and the parameter \( \eta > 0 \) is referred to as the learning rate. The function used to compute the loss gradient can vary widely between models. Common loss functions include regression loss and cross entropy classification loss [14]. After each update, the gradient is reevaluated and the process is repeated. Note that the error is always computed on the training data. With the loss defined, backprop calculates the gradient of the loss with respect to the parameters of the network [34]. An example of backpropagation is illustrated in Figure 2.2.4. From there, the error associated with each neuron can be computed. An example calculation is shown below:

\[ \partial_n = h'(a_n) \sum_{k=1}^{K} w_{kn} \partial_k \]  

(8)

where \( \partial \) represents the error associated with a particular neuron. The backpropagation algorithm can be applied to any network with any number of outputs by application of the chain rule and power rule [34].
Figure 2.2.4: This figure shows an illustration of the calculation of error terms \( \delta \) associated with each weight by backpropagation of the \( \delta' \)s from neurons in the output layer \( k \) which has connections with neuron \( n \). The green arrow denotes the direction of information flow through forward propagation. The red arrows denote the direction of error information in backward propagation in the hidden layers associated with the neuron \( z^l_m \).

### 2.2.3. Common Models

In this section we review some common CNN models that are used in chapters 4 and 5 of this work. CNN object detection models have a DNN feature extractor ‘backbone’ and an object detecting ‘head’ [30]. Information flows through the CNN from the backbone to the head. The backbone of a model extracts local visual features from an input image. The head makes predictions about the location and class of a potential
object based on the features extracted from the backbone. An example of a CNN model is shown in Figure 2.2.5.

![CNN Architecture](image)

**Figure 2.2.5:** This figure shows an illustration of a CNN architecture for classification. Repeated blocks of multiple convolutional and pooling layers extract features from an input image. These features are fed to a series of fully connected layers. The final classification decision of the network is the maximum output of the last fully connected layer. From [https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53](https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53).

**ResNet** (Residual Network) is a classification neural network that is commonly used as a backbone in many CNN models [8]. There are many variations of the ResNet, but the most common is ResNet50 [8]. ResNet50 is comprised of 34 layers of interleaved convolutional layers and max pooling layers with the final layers being fully connected layers. In this work we use a Darknet 53 backbone, which is similar to ResNet [35]. This backbone was chosen because it is the standard backbone of YOLO detection models [35] [36] [37], which allows for easier comparison to other models in the literature.

**YOLOV3** is the detection head used in this work. YOLOV3, henceforth referred to as YOLO (You Only Look Once), is a representative example of a one-stage object
detection model which tries to both localize and classify an object simultaneously [38] [39], which is desirable for real time operability that remote sensing tasks often require. For an object localization task, a model must draw a bounding box around one or more objects in an image. For a classification task, the model must predict a class label and a corresponding class probability for those objects.

YOLO divides the input image into an $S \times S$ grid such that each cell of the grid is a fixed 32 x 32 pixels square. A grid cell can only be associated with one object, and it predicts a fixed number of bounding boxes, $B$. To calculate the right bounding box for a grid cell, it uses the one with the highest (IoU) linking criteria with the ground truth data. The loss function of a YOLO detector is the sum of localization (box regression), objectness loss, and classification loss. It is of the form

$$L = L_{box} + L_{obj} + L_{cls}$$  \hspace{1cm} (9)

To predict a bounding box YOLO uses anchor boxes. Anchor boxes are predefined boxes of specified size and the positions of which are defined as offsets from the grid cell locations. YOLO predicts bounding boxes as adjustments to the anchors via linear regression of anchor box coordinates. Objectness loss represents the probability that some object exists within a given box, it is bounded between [0,1]. The classification loss represents the conditional probability of a class given that there is an object in a box, which is also bounded by [0,1]. The final predicted information for each box is 5-
dimensional tensor \((x, y, w, h, C)\), where \([x, y, w, h]\) represents the box coordinates, and \(C\), is the total confidence score for a prediction, which is the product of objectness and class confidence scores [39].

2.3. Image Stylization

In the work presented in chapters 4 and 5 we are concerned with style transfer that we proposed to use in order to improve performance of RGB pretrained detection models on IR object detection tasks. We hypothesize that he more similar in appearance existing RGB images can be made to IR images, the more valuable they are as training examples to a deep IR object detector which will ultimately lead to greater detection performance. We seek to learn a mapping, or stylization, from RGB images to IR images. This is an application of the image stylization task [40].

Image stylization is the general task of altering images to adopt the visual appearance or ‘style’ of another image or images. Modern image stylization algorithms are characterized by their use of DNNs and are generally referred to as neural style transfer (NST) algorithms. NST algorithms were first proposed by Gatys et al. in [26]. The general mechanism used by NST models is to align the source and target modality distributions extracted from imagery in each modality with a neural network at either the feature or pixel level. Alignment is achieved in one of two ways: minimizing a distance metric, or equivalently through use of an adversarial critic [40].
NST models that use adversarial critics are known as adversarial NST (ANST). ANST models achieve stylization by training a stylizer (or generator) network to create an image to maximize the error of an adversarial discriminator (or critic) that is trained to distinguish between the source and target modalities [41] (i.e., the ideal generator creates a synthetic image that looks so realistic that a DNN cannot distinguish the synthetic image from real images) [41]. NST models that minimize a distance metric and do not use an adversarial critic are known as photorealistic NST (PNST) models.

ANST models are based on the technique of adversarial learning [41]. In the simplest adversarial scenario, two networks contest each other. In this contest, one network’s gain is another network’s loss. One network is a generative model, or ‘generator’, that creates synthetic images either by generating novel images from noise or operating on real images in one domain to create a synthetic image in another domain (e.g., different sensing modalities). These synthetic images (along with real images) are then used to train a classification network, or ‘discriminator’. The generator learns to create convincing synthetic data that is similar to real data. The discriminator learns to distinguish candidates produced by the generator from the true data. The training objective of the generator is to increase the error of the discriminator. The training objective of the discriminator is to minimize classification error when deciding if a proposal is from real data or data fabricated by the generator. This forces the generator to learn a robust stylization to “fool” the discriminator.
ANST models have the powerful ability to abstract features resulting in highly complex stylizations; but they tend to produce more ‘artistic’ stylization that distorts the semantic content of images in contrast to the photorealism of PNSTs (e.g., adding ‘brushstrokes’ to images in Photo→Portrait style transfer) [42]. ANST models are related to generative adversarial networks (GANs) proposed by Goodfellow et al. [43]. Previous work has applied GANs to image stylization [44] by directly converting a source modality image into a target modality images [44]. However, this requires paired images that are highly correlated (i.e., show the same scene or objects with similar attributes) for training. However, paired training data is often not available in RGB→IR stylization problems, which are the focus of this work (e.g., [45]).

There are several ANST models where paired data is not required. Among ANST models, the most well cited and used are cycle-consistent models [40]. CycleGAN, first proposed by Zhu et al. [46], was designed to eliminate GAN’s reliance on paired data. CycleGAN is reviewed in subsection 2.3.1. Later work that was based on this approach is motivated by the effectiveness of the cycle consistent methods. Liu et al. concurrently with CycleGAN proposed UNIT which assumes the existence of a shared latent space between any two images in separate modalities with added cycle consistency [47]. Later CyCADA, proposed by Hoffman et al. [48], further improved on CycleGAN by adding semantic consistency objectives to the model as well as a task objective. Semantic
consistency incentivizes the preservation of semantic information; task loss is an additional objective that takes advantage of source/target modality labels. The use of task loss gives CyCADA access to more information when learning a stylization function. CyCADA is reviewed in subsection 2.3.2.

**PNST** models are primarily concerned with preserving the semantic content in stylized imagery. Luan et al. proposed deep photo style transfer (DPST) [42] which adds a regularization objective to the standard NST alignment algorithm and uses semantic segmentation masks in stylization. But DPST requires heavy computation. State-of-the-art PNST algorithms are extensions of the whitening-color-transform (WCT) algorithm [49]. WCT applies a linear transform that maximizes the covariance of features extracted from content image and style image pairs [50]. Li et al. [51] proposed Photo-WCT which mitigates spatial distortions in stylized imagery by providing pooling masks to the decoder. This accounts for loss in information as images are encoded in feature space. However, the visual output of Photo-WCT itself is unrealistically warp and blur an image, losing information about fine details in an image; the authors employ post processing smoothing and filtering operations to mitigate the problems with Photo-WCT [51]. This post processing erroneously blurs the final output and adds computational complexity. Yoo et al. [49] proposed WCT2, which like Photo-WCT was motivated to mitigate unrealistic artifacts in stylized images to due information loss in the encoding process. WCT2 is reviewed in subsection 2.3.3.
2.3.1. CycleGAN

In our work CycleGAN is used as a representative baseline ANST model and its performance is compared to those we propose. One branch of the architecture of CycleGAN is illustrated in Figure 2.3.1.

CycleGAN has two generator and discriminator sets. The first generator ($G_{XY}$) maps real images in domain $X$ to domain $Y$. The first discriminator ($D_Y$) then evaluates if the new image is fake, or from the distribution of real domain $Y$ images. Then, a second generator ($G_{YX}$) maps the stylized image from domain $Y$ back to the original domain $X$. A second discriminator ($D_X$ (not pictured)) then evaluates if the new reconstructed image is fake or from the distribution of real domain $X$ images. The difference between the original image and reconstructed image is then measured ($L_{cyx}$), this is known as cycle consistency loss. This cycle of domain transfer allows for CycleGAN to approximate paired data in each domain [46]. Cycle consistency loss is illustrated in Figure 2.3.2.
Figure 2.3.1: This figure shows the architecture of the CycleGAN, for simplicity only the XY branch is shown. A source image in domain X is fed to a generator, $G_{XY}$, which learns a mapping from domain X to domain Y. A discriminator, $D_Y$, then attempts to classify if the image stylized by the generator is a real image from domain Y or not. Next the source image stylized in domain Y is fed to a second generator $G_{YX}$ which maps the image from domain Y back to the original domain X.

Cycle consistency loss is computed based on the difference between the reconstruction and the source image as detailed in Figure 2.3.2. The $G_{XY}$ and $D_Y$ models are trained jointly with two separate loss terms: Gan loss and Cycle Consistency loss. For source images in the Y domain (not pictured), a separate discriminator exists for images stylized and reconstructed with $G_{YX}$ and $G_{XY}$ respectively.
Figure 2.3.2: This figure is an illustration of one direction of cycle consistency loss. Image x from domain X is stylized by the generator $G_{XY}$, which produces a stylized image $G_{XY}(x)$ in the Y domain. The resulting image is then stylized back to the original domain X by $G_{YX}$ to create a reconstruction image $G_{YX}(G_{XY}(x))$. The cycle consistency loss is the L1 difference between the reconstructed and the original image.

The training objective of CycleGAN is to minimize loss, the loss function of CycleGAN is the weighted sum of the classification loss of each discriminator, and the cycle consistency loss of each generator [46]. It is of the form:

$$L = L_{cycx} + L_{DY} + L_{cycy} + L_{DX}$$

(10)

where $L_{DX}$, and $L_{DY}$ are the classification losses (section 2.1.4) of the discriminators, and $L_{cycx}$, and $L_{cycy}$ are the cycle consistency losses for the $G_{XY}$ and $G_{YX}$ generators respectively [46]. However, cycle consistency does not guarantee that the stylization learned by a generator will be ‘good’, only that it is invertible. It is a well-known
problem that CycleGAN may unrealistically warp stylized images, or erroneously remove semantic content from an image (even insert new semantic content into an image) [46]. Despite this, CycleGAN is used widely for image stylization tasks [46] [40].

2.3.2. CyCADA

CyCADA is an extension of CycleGAN [48] designed to better preserve spatial statistics of the stylized imagery and increase the robustness of the stylization. We use CyCADA in this work as a representative example of state-of-the-art ANST models. The overall architecture is the same as CycleGAN (2.3.1) with two key additions to train the generative part of the model: a semantic consistency loss between the source and stylized image (i.e., an additional cycle consistency loss), and a small task loss which makes use of loose labels in the source domain as extra training information compared to CycleGAN. Specifically, the task loss is a classification task where classification of a small classifier ‘task network’ serves as the loss. This task loss helps increase the robustness of stylization such that CyCADA learns an (ideally) class conditional stylization. A diagram of CyCADA is shown in Figure 2.3.3.
Figure 2.3.3: This figure shows an illustration of CyCADA. The architecture is the same as CycleGAN but with additional semantic consistency and task loss terms (gray arrows).

### 2.3.3. WCT2

WCT2 is used in this work as a representative example of a state-of-the-art PNST models. The WCT algorithm is illustrated in Figure 2.3.4. The input to WCT is a pair of images: a content image $I_c$ and a style image $I_s$ that are used to create an output image $I$. The content of an image is encoded in feature space and then mapped into a learned color space (the color space of the style image). The mapping is based on matching the covariance of the features extracted from $I_c$ and $I_s$. The transformation applied to the features to match covariance is linear and thus does not warp or alter the image in an
un-photorealistic way. The linearity of the transform has the added benefit of explicitly preserving information and the spatial statistics of features. Once matched, the stylized features are then decoded back to image space.

Figure 2.3.4: This figure provides an overview of the operation of the WCT algorithm. A content image and a style image $I_c$ and $I_s$ are encoded in feature space using an identical encoder, thus the equals sign. The content features $f_c$ are whitened with a linear transform which projects the image into a ‘content domain’ where the covariance of the whitened features is identity, and the features have no style. Another linear transform colors the whitened features $f_w$ to match the covariance of the style features $f_s$. The colored features $f$ are then decoded back to image space.

The main innovations of WCT2 over similar algorithms are a superior encoder/decoder architecture which has both multiple WCT transforms and replaces
max pooling layers with Fourier pooling layers [49]. Multiple WCT transforms help increase the robustness of the stylization [49]. Fourier pooling is a version of max pooling which preserves spatial information, and therefore image content [49]. The WCT operation is a fixed transform which is calculated based on matching the covariance of images in feature space [50]. Therefore, the WCT2 model does not need to be trained to perform stylization outside of pretraining the encoder/decoder. The authors also use multiple WCT projections throughout their encoder for a more robust stylization.

### 2.4. Automatic Augmentation

NST algorithms are not the only way to achieve image stylization for our transfer learning goals. In chapter 5 we present a novel image stylization method using automatic augmentation methods that will be reviewed in this section. Data augmentation applies a composition of functions to data which alters them slightly and creates technically novel training examples for a deep model [52]. This practice allows for more efficient use of training data. There are many common methods for augmentation. Indeed, augmentation is a virtually ubiquitous machine learning practice [53]. However, the past augmentation policies are hand crafted and may not be ideal across multiple applications.

In recent years, researchers have developed methods for data driven inference of good augmentations to apply to data, this is known as Automatic Augmentation. Automatic Augmentation uses meta-learning to cast the selection of an augmentation
policy as differentiable, and therefore learnable with standard gradient descent methods [27]. In our work, we hypothesize that a composition of nonlinear functions could be used for image stylization. With automatic augmentation, we can learn this composition directly.

Several approaches for automatic augmentation have been developed. The first of these was AutoAugment (AA) [27]. In the AA algorithm there are two models: A Controller and a Child model. The Controller model controls what policy is learned. A policy controls what operations are applied to an image, the probability of an operation being applied, and the magnitude of the operation. Each of these parameters is learnable. The policy learned by the Controller is then used to train a downstream Child model. In [27], the downstream task is classification on ImageNet. The validation accuracy of the Child model is then used as a reward signal to train the Controller model via reinforcement learning [54]. A diagram of this algorithm is shown in Figure 2.4.1.

Figure 2.4.1: This figure illustrates the optimization strategy of AutoAugment. A controller model samples an augmentation policy $S$. This policy dictates what operations are used to augment image, the probability of each operation being
applied, and the magnitude of each operation. A Child network is trained using data augmented by policy $S$ on a separate downstream task (e.g., ImageNet classification). The validation accuracy of the child model, $R$, is used as a reward signal to update the controller via reinforcement learning.

It is worth emphasizing that a single update to the Controller requires the entire end-to-end training of a Child network, which is extremely computationally inefficient. In fact, it took nearly two years of GPU compute time to learn an ideal augmentation policy for ImageNet [27].

Shortly after the publication of AA, many papers were published on work that tried to search for a policy more efficiently [55]. In this work, we use Faster AutoAugment (Faster AA) to learn an augmentation policy [55]. The architecture of Faster AA is shown in Figure 2.4.2. The general idea of Faster AA is to use an adversarial critic to guide policy selection. By using an adversarial critic, Faster AA does not require training a Child model for each controller update. What required two years of compute time for AA, [55] was achieved in 40 minutes of compute time by Faster AA. It is for this reason we chose to adapt Faster AA to our research.
Figure 2.4.2: This figure provides an overview of Faster AutoAugment algorithm. Data is augmented by a controller model which selects an augmentation policy. The augmented data and the original data are then fed to an adversarial critic which attempts to correctly guess if an image is an original, or an augmented image. A classification loss term is added to help regularize the policy generator. The policy is updated by backpropagation based on the decision of the critic, which is much faster and more efficient than the original AutoAugment model. Adapted from [55].

The training objective of Faster AA is to match the distributions of statistics between augmented data and original, un-augmented, data while preserving classification accuracy. The loss function is of the form:

$$L = L_{adv} + L_{cls}$$  \hspace{1cm} (11)
where $L_{cls}$ is the cross-entropy classification loss, $L_{adv}$ represents the average critic decision statistics computed over a batch of original and augmented image pairs. The classification loss has a regularizing effect on the policy selection and provides the critic with more information while learning a given policy compared to NST algorithms. The form of $L_{adv}$ is

$$L_{adv} = \frac{1}{b} \sum_{i}^{b} (C(x_i^*) - C(x_i))$$

(12)

where $C(x_i)$ represents the critic decision statistic on augmented data $x_i$ representing the $i^{th}$ data sample in batch of $b$ samples, and $C(x_i^*)$ represents the critic’s decision statistic on augmented data $x_i^*$. By minimizing these objectives, Faster AA learns an augmentation policy which fills gaps in and around this distribution. This concept is illustrated in Figure 2.4.3. Augmenting with a learned policy allows for a downstream model to more richly sample a given data distribution which tends to be yield better performance on tasks for that dataset [55].
Figure 2.4.3: This figure illustrates the objective of AutoAugment. The authors cast data augmentation as a process that fills missing points in the original distribution of training data. Therefore, the training objective is to minimize the distance between the distributions of features extracted from the augmented data (red triangles) and features of the original data (blue circles). Adapted from [55].

2.5. Summary

This chapter reviewed the background material necessary to develop the methods proposed in subsequent chapters of this work. An introduction was given to the basic methodologies of algorithm development for object classification, detection, NST, and Automatic Augmentation tasks. These conventions are used throughout the remainder of the document to quantify and compare the performance of proposed algorithms to the performance of existing algorithms. The next chapter begins describing our work on developing buried threat detection algorithms for use on data from handheld GPR systems.
3. Buried Threat Detection

This chapter discusses our work developing BTD algorithms for landmine detections. We aim to develop algorithms that improve the classification accuracy of statistical models for hand-held landmine detection. To this end our strategy is to leverage large, preexisting downward looking (vehicle based) landmine detection datasets to train hand-held detection algorithms. The contributions in this chapter can be summarized as follows:

1. Leveraging large quantities of downward-looking GPR data to improve landmine ATR algorithms for hand-held GPR. Based upon this work we found that simple pre-processing of downward-looking GPR makes it suitable to train machine learning models that are effective at recognizing threats in hand-held GPR as well. This leverages many decades of data collection performed on downward-looking GPR to mitigate data collection for hand-held systems.

3.1. Introduction to BTD

Buried Threat Detection (BTD) is the task of detecting and identifying buried threats (e.g., landmines, unexploded explosive ordnance, etc.) with a remote sensor. There are an estimated 110 million landmines distributed across Earth [56]. This contamination kills or maims thousands of people each year, costs hundreds of dollars
to remove each individual threat, and ultimately renders hundreds of thousands square kilometers of land unusable [56].

Ground penetrating radar (GPR) is a sensing modality that has been investigated extensively for BTD [57] [58] [59] [60] [1]. Much of this research focuses on downward looking (or ‘vehicle mounted’) GPR (DL-GPR) for route clearance. DL-GPR systems typically consist of a vehicle mounted array of antennas that collect GPR data as the vehicle moves down a road or open area. Hand-held GPR systems (HH-GPR) have also been explored for use by dismounted personnel, and these systems typically consist of a small hand-held apparatus that can be operated by a single user on foot [61] [23]. The small form-factor of HH-GPR systems allows the ability to operate off road and in terrain that is inaccessible to vehicles.

One of the biggest challenges in developing detection algorithms for HH-GPR is a general scarcity of labeled data [24]. In contrast to vehicle-mounted DL-GPR systems, HH-GPR data must be collected manually, on foot [24]. Data is an important resource for the development of modern BTD algorithms, most of which rely on supervised classification models [1]. These models must be trained to distinguish between GPR data corresponding to threats and non-threats, respectively, using labeled examples. Supervised models tend to perform more poorly when data is scarce, and it is difficult to accurately estimate their performance in the field using small training datasets [62].
In this work we consider leveraging abundant labeled DL-GPR data to supplement the training data used for supervised HH-GPR algorithms. The basic sensing mechanism underlying HH-GPR and DL-GPR systems are very similar; in both cases a GPR sensor emits a pulse into the ground, and subsequently measures the radar signals reflected to the antenna from the subsurface. GPR data can be concatenated to form imagery known as B-scans, on which detection can be performed on both HH-GPR and DL-GPR [24]. However, DL systems are used to detect anti-tank (AT) landmines as opposed to HH-GPR systems which are used to detect anti-personnel (AP) and AT mines. However, AT and AP mines have the same general physical shape and therefore have similar hyperbolic target like signatures that characterize targets in GPR data [63]. Therefore, we hypothesize that given these similarities we can stylize data from AT targets by strategically down sampling them to look like data from AP targets and increase the available training data of both types of targets and that this increased availability of data will yield more accurate classification of targets. This approach leverages decades of data collection performed with DL-GPR systems, and potentially expedites the development of successful HH-GPR-based BTD algorithms. Our methods for accomplishing this are discussed later in this chapter.

3.2. Typical BTD Processing

Often raw sensor data is very large, and it is not feasible to apply a sophisticated classification algorithm to all of it in a real time operational setting (e.g., as an operator
moves down a minefield with a sensor) to distinguish targets from clutter (e.g., shrapnel, cans, coins, roots, rocks, etc.). Because of this, BTD is often divided into two distinct processing steps: search processing followed by region processing (e.g. [1], [64]). This process is outlined in Figure 3.2.1. The exact methods of data collection are outlined in section 3.4. Prescreening or ‘search processing’ applies a computationally inexpensive algorithm to search the total population of data quickly and efficiently to find locations that are likely to contain a target. A spot is likely to contain a target if the prescreener measures a high energy signal due to the reflection of radar due to objects in the ground. However, GPR sensors also respond to clutter; any dielectric interface in the ground due to either targets or clutter will cause a reflection of the radar to the sensor that yields many false alarms due to clutter. This is by design, as a prescreener should have a high likelihood of identifying most (ideally all) true targets due to the potentially mortal consequences of a single missed target. More advanced processing (i.e., feature-based processing) can then be applied to the specific locations marked by the prescreener as being potentially important. This combined with prescreening dramatically reduces the overall computational cost compared to using a feature-based processing algorithm on the entirety of the sensor data.

It is the goal of the feature processing or ‘region processing’ step to perform more advanced processing and reduce the number of false alarms that arise from clutter, as it would be untenable to dig up every location flagged by the prescreener. Region
processing typically consists of extracting local statistics or ‘features’ from data around each prescreener alarm location and then applying a classification algorithm to assign a detection statistic (e.g., a number indicative of ‘target’ or ‘clutter’) to that location. The feature processing only operates on prescreener alarms; therefore, it cannot improve the detection probability of the entire BTD system. If a target was missed by the prescreener, it will not be considered by the feature processing algorithm. However, feature processing can dramatically reduce the number of false alarms while missing very few of the true targets [65].

![Typical BTR Detection Processing](image)

**Figure 3.2.1:** This figure shows a flowchart of a common BTD processing flow. Data from a sensor, often in the form of a volume or an area, is fed to a prescreener (left). Prescreeners consist of a computationally fast detection algorithm that identifies individual locations in the data volume that are likely to contain a buried object, i.e., “alarms”. An example B-scan is shown (center) with a target signal marked in the black square. Prescreening reduces the amount of data that must be considered in the next step, feature processing, which is more computationally intensive but often yields better discrimination between true buried objects and false alarms. Feature processing (right) typically consists of extraction of a variety of local statistics around each alarm location and using a classification algorithm to assign a decision statistic based on the data statistics.
3.3. Evaluating BTD Algorithms

Algorithms used for BTD must localize and classify a set of features. As described in section 2.1.2, predictions from detection algorithms are first ‘linked’ with a linking function that compares the predictions of the algorithm to labeled ground truth. The linking function used in this work is referred to as a ‘halo’ where a prediction is considered a hit if the prediction is within some radius (in pixels) of an alarm location. An illustration of BTD linking can be seen in Figure 3.3.1. Once the predictions of the algorithm have been linked, we compute ROC curves to fully characterize the performance of the algorithm as defined in chapter 2.

![Example Detector Output](image)

Figure 3.3.1: This figure provides an illustration of how detector alarms are a label for scoring by using a halo-based linking criteria. The detection system output for a typical experiment consists of a set of locations on the earth, one for each alarm made by the system. These alarms are shown with blue triangles. Each alarm must be labeled as a hit or a false alarm for scoring, indicating whether it refers to a true target or not. The location of each buried object in the testing area is known in advance, and any alarm landing within a predefined radius, or halo, around a target is considered a hit. The target locations and their halos are shown in red on the figure.
All alarms that occur outside any halo are considered false alarms, as shown. If multiple hits are declared on a single target only the hit with the maximum decision statistic is evaluated in scoring. In this example, there are a total of 3 hits, 1 false alarm, and one miss, which are labeled in the figure.

### 3.4. GPR Data

In this work we are concerned with processing data collected by a HH-GPR sensor, where data is collected at regular intervals as the operator swings the sensor side-to-side, perpendicular to the direction of travel. Labeled data from HH landmine detection systems are generally collected in training lanes using a push button or other marker in the system to mark in the data when the sensor passes over a target or clutter at known locations. Data is collected over a set of inert targets where an operator is free to place a sensor directly over a threat object and can ignore limiting factors such as sensor lag. Data collected under these conditions represents the best-case scenario for a detection algorithm. From there, practical BTD algorithms or sensor operating methods can be simulated which can then be deployed in a live minefield environment such that they pose minimal risk to an operator while maintaining sufficiently high classification accuracy, though such deployment considerations are not the focus of this work.
Figure 3.4.1: This figure shows an example of a typical sweep pattern for a HH-GPR sensor, i.e., the spatial location of the antennas of the HH-GPR as it is swung by an operator over the ground along the direction of travel (approximately along Spatial axis 1). A B-scan, indicated in green, the location of which was marked by the operator, is then extracted around the alarm location which is indicated with a black dot. For each B-scan, each pixel is a measurement of the amplitude of the subsurface reflection at a particular time and location. The vertical axis represents time to measure the reflection and is correlated with depth. The horizontal axis represents the spatial location of the radar position over the surface of the ground.

The HH-GPR system employed in this work is an ultra-wideband impulse-based ground penetrating radar with two data channels, where each channel represents data collected by one of two dipole antennas. Since there are two radar channels, there are two co-located B-scans collected in parallel as the HH-GPR sweeps. During data collection, the HH-GPR is held such that the radar aperture itself is a short distance (on the order of a meter) in front of an operator and such that the antennas are pointed perpendicularly to the ground. Data is collected as the operator travels down the path and swings the radar aperture in a semi regular side-to-side pattern as shown in Figure
3.4.1 (left) along with a corresponding B-scan at a target location (right). This system, or similar ones, have been used in several previous studies [1]. The HH-GPR system was used to collect multiple runs of data over 12 distinct test lanes, resulting in a GPR dataset comprising a total area of 131.21 m². For this data collection, 65 unique threats buried at various depths were used, resulting in a total of 130 threat encounters.

The DL-GPR system used in this work is comprised of an array of antennas which is attached to the front of a vehicle [66]. The array is arranged so that it is perpendicular to the direction of travel of the vehicle, and each individual antenna is pointed down towards the ground. Similar to the HH-GPR, each antenna emits an ultra-wideband radar signal, consisting of a differentiated Gaussian pulse, and then measures the energy reflected back towards the array from the subsurface. Data is collected as the vehicle travels down a road, path, or lane, which results in a 3-D data cube with two spatial dimensions (across the panel, and along the direction of travel) and one temporal dimension, into the ground. In this study the GPR system was used to collect data multiple times over 13 distinct lanes at two U.S. test sites. The final DL-GPR dataset comprised a total scan area of approximately 120,000 m². A total of 664 unique buried threats were emplaced in the testing lanes, resulting in a total of 4,552 threat encounters in the dataset.
3.4.1. GPR Data Features

In this section, we describe the gradient-histogram (GH) feature that we employed as a baseline in our experiments. Our GH feature is modeled largely after the EHD feature [67] and GPR-HOG (histogram of oriented gradients) feature from DL-GPR community [65]. The premise of the GH feature is to split the image into disjoint pooling regions (PRs) and then succinctly summarize the shape information within each PR. This is accomplished by creating histograms of the dominant gradient orientations in each cell. An example of GH feature computation for each PR is illustrated in Figure 3.4.2.

Our GH approach begins with a series of filtering operations (i.e., convolutions) designed to estimate the dominant gradient magnitude, $G$, and its orientation, $\theta$, at each pixel location in an input B-scan, denoted $I$. Following the derivation of the EHD feature, we employ filters $h_{\theta_k}$ to identify four possible dominant angles, given by $\theta_k = (k - 1)(\pi/4), k \in \{1, 2, 3, 4\}$. Sobel filters rotated to the desired angles were used for this application. Given an input B-scan we compute the gradient magnitude for a desired $\theta_k$ by:

$$G_x = I \ast h_x, \quad G_y = I \ast h_y. \quad (13)$$

The magnitude of the gradient at each pixel can be calculated as:

$$G(i,j) = \sqrt{G_x(i,j)^2 + G_y(i,j)^2} \quad (14)$$
The dominant gradient angle at each pixel is calculated as:

$$\theta(i, j) = \tan^{-1}\left(\frac{G_y(i, j)}{G_x(i, j)}\right).$$  \hspace{1cm} (15)

Consider $n_\theta$ angle bins between 0 and $\pi$ degrees. Let the edges of the $n_\theta$ bins correspond to $n_\theta + 1$ edge values $\phi_k = 180k/n_\theta$, where $k = 0, 1, ..., n_\theta$. GH features describe the local statistics of the image gradients by allowing each pixel to be a vote for a specific angle bin, with a vote magnitude proportional to the gradient magnitude at each location. We define a 3-dimensional matrix that records the dominant gradient orientation, and its corresponding magnitude, at each pixel location. Mathematically, this matrix is defined as follows:

$$V(i, j, k) = G(i, j) \delta(\phi_{k-1} < \theta(i, j) \leq \phi_k) \delta(G(i, j) > \lambda).$$  \hspace{1cm} (16)

Here $\delta(x)$ takes the value 1 when the input argument is true, and 0 when it is false. In equation (3), if no gradient exceeds a threshold value $\lambda$, it is assigned to an extra ‘no angle’ bin in the matrix given by:

$$V(i, j, n_\theta + 1) = G(i, j)\delta(G(i, j) \leq \lambda)$$  \hspace{1cm} (17)

The next computation involves aggregating the gradients within each PR. Once pooled, each region yields a single histogram of the dominant gradients within it. Let $c$
denote the pixels in a pooling region, then we compute the histogram for that pooling region by:

\[ H(k) = \sum_{(i,j) \in c} V(i, j, k) \]  

(18)

To compute the final GH feature, we concatenate the histograms from each PR.

---

**Figure 3.4.2:** This figure shows an illustration of the computation for our gradient visual feature. Left: GPR image data is extracted from each pooling region (black squares). Center: The gradients from each pixel in a pooling region are computed and binned in a histogram. In this illustration, and in our experiments, five angle bins are employed. Right: The histograms for each pooling region are concatenated to create a final feature vector describing the input image.

### 3.4.2. Model Implementation

In this work we use a support vector machine (SVM) classifier with a radial basis function kernel. The SVM is a representative non-linear classifier which is well established in the BTD literature [21] [68]. The choice of SVM in this work allows us to compare to previous works more directly. The SVM is a supervised nonlinear classifier, which means it must be trained to distinguish between threats and non-threats using
labeled examples of data from each class. It is trained to learn a discriminative function which maximizes the margins between the distributions of target and nontarget statistics [25]. Supervised classifiers have been shown to have excellent performance on GPR-based BTD [1].

Once trained, the SVM can be used to predict scores for unlabeled data. A typical approach for obtaining training data is to extract a B-scans from GPR imagery at an alarm location. Although the prescreener provides the spatial location of the alarms, it does not localize them in time. To address this need, we used the Maximum Smoothed Energy Keypoint (MSEK) localization method [62], which uses signal energy as a cue to identify temporal locations at which sub-patches can be extracted. We used MSEK to identify four sub-images at each alarm location to use for training our supervised classifiers. As discussed, once it is trained, a classifier can be used to provide scores at new locations. During score inference, we follow the procedure described in Reichman et al. 2017 [62]; the classifier is applied to densely extracted sub-images along the temporal axis at each alarm location. A final score is computed for an alarm by the summing top $L = 6$ highest scores.

3.5. Experiments

In this section, we describe our experimental design and results. In each experiment we perform object-based cross-validation of our HH-GPR BTD algorithm. In each case we augment the HH-GPR data used to train the supervised classifiers with a
supplementary dataset of DL-GPR training data, and we vary the quantity and characteristics of the supplementary data in each case. This experimental design is illustrated in Figure 3.5.1. The quantity and characteristics of the data are controlled using several experimental variables, which are illustrated and explained in Figure 3.5.2. Although we supplement with DL-GPR data, we only score the algorithms on HH-GPR data, since our ultimate goal is to develop more accurate HH-GPR-based BTD algorithms.

Figure 3.5.1: This figure is a diagram of our experimental design. DL-GPR data is sub sampled based on a selection criteria to create a supplementary dataset. We use the supplementary dataset and a HH-GPR training set to train a classifier model. The trained classifier (blue) is then evaluated on HH-GPR data. The vector of decision statistics of the trained model (gray) are then linked, scored, and we assess the performance of the model.

We hypothesize data supplementation is beneficial, but how best to optimize the benefit of the additional data needs to be explored. For instance, how much and what type of DL-GPR data should be used (i.e., threats, nonthreats, or both) needs to be
explored in a data-driven way. Our experiments are divided into three sections: supplementation with threat alarms (section 3.5.1), supplementation with non-threat alarms (section 3.5.2) and supplementation with both threat and non-threat alarms (section 3.5.3).

The process by which we construct a supplementary dataset is shown in Figure 3.5.2. We begin by spatially resampling the DL-GPR data. We assume DL-GPR contains a subset of the threat objects represented in HH-GPR. In particular, DL-GPR contains data from anti-tank (AT) mines, which are larger than the anti-personnel (AP) mines that are common in HH-GPR data. AT and AP mines however have the same general shape and appear with the same characteristic hyperbolic threat signature in GPR data. Therefore, we spatially downsample (or stylize) the DL-GPR data such that target objects appear to have the same spatial extent. The rate of down sampling was estimated using knowledge of the size of typical AT/AP mines, and the spatial resolution of DL-GPR and HH-GPR radars. From there we estimate how many centimeters correspond to a pixel of DL-GPR data and HH-GPR data, and downsample by a factor \( r \) data so that the spatial extent of targets in DL-GPR data corresponds those in HH-GPR data. One we estimated a range of \( r \), we did an experiment where we trained several classifiers using DL-GPR and HH-GPR data measuring performance as a function of \( r \). We observed that the resampling rate of \( r = 0.1 \) was ideal, we use this resampling rate in our experiments. We then select what category of DL data we wish to
use in our supplementation dataset and filter our data from the total population of threat objects, false alarms, or both based on the decisions of the prescreener. We assume that the ‘quality’ of DL-GPR data used is important i.e., an alarm that corresponds to a B-scan with a pronounced target-like signal is a more valuable training example than an alarm corresponding to a B-scan that looks like noise. We use the prescreener confidence statistic as an estimate of DL-GPR data ‘quality’. With this hypothesis in mind, we filter DL-GPR data further by applying a threshold to the confidence score, $P$, of the prescreener described in section 3.4.2. A higher confidence threat identified the prescreener suggests a more obvious target signal. Finally, we randomly select $N\%$ of DL-GPR data from the remaining candidates, where $N$ is a percentage of the size of our HH-GPR training dataset. We repeat this sampling procedure for each of our experiments.
Figure 3.5.2: This figure provides an illustration of the procedure for creating the supplementary training dataset. We first resample the DL data spatially. Then we divide the total DL dataset into threat, non-threat, and combined (i.e., all data) subsets. We exclude all alarms outside of a prescreener confidence percentile range, given by $P = [p_L, p_H]$% percentile range. Of the remaining alarms, we randomly sample $N\%$ of the alarms to include in the supplementary dataset, where $N$ is given as a proportion of the total number of HH-GPR training alarms.
We conduct controlled experiments to measure the effectiveness of data supplementation by sweeping over the parameters of our sampling algorithm \( P \) and \( N \) for each type of DL-GPR alarm (i.e., threats (T), nonthreats (NT), both). If the quality of the DL-GPR data does affect performance we assume that this will be most apparent and beneficial at the extrema of quality i.e., the highest and lowest quality data. We therefore sampled in two directions: ‘from the top’ where we sample from the alarms with the highest prescreener scores (fixing \( p_H \)) and gradually allow lower quality alarms into the training data by sweeping \( p_L \), and ‘from the bottom’ where we sample the lowest confidence (fixing \( p_L \)) alarms and gradually allow higher quality alarms into the training data by sweeping \( p_H \). We then measure the pAUC up to a FAR of 0.5 as a performance metric. In section 3.5.3 we create a supplementary dataset comprised of both threat and non-threat alarms, using the best configurations found in sections 3.5.1 and 3.5.2 respectively. Our aim in the last experiment is to identify whether the threat and non-threat supplementary dataset are complementary, and thereby to maximize performance.

### 3.5.1. Data Supplementation With Target Data

In this section we present our results investigating supplementation with DL-GPR data collected over threats. We begin by varying the size of the supplementation dataset (\( N\% \)), and sampling the threat alarms uniformly from the total population of DL-GPR prescreener alarms (i.e., \( P = [0,100] \)). The results of these experiments are
presented in Table 3.5.1. The results indicate that supplementation with any quantity of DL-GPR threat data improves detection performance. Performance improved as $N$ increased and seemed to saturate around $N\% = 15 - 20$, and therefore we stopped sampling.

Once we established an appropriate value for $N$, we varied the percentile range of prescreener scores, $P$, from which the threat supplementary dataset was constructed. The results of these experiments are presented in Table 3.5.2, where $N\% = 20$ for all experiments. As we hypothesized, sampling only among the top 25% of threats (i.e., $P = [75 - 100]$) achieves better results than sampling over the entire population. Threats in this range exhibit the canonical hyperbolic pattern of buried threats, suggesting that it is possible that the best strategy for developing a classifier for processing HH-GPR data is to focus training on identifying hyperbolic patterns. The results indicate that sampling only lower-confidence threats tends to result in poorer performance compared to sampling over the entire threat population uniformly.
Table 3.5.1: Performance results when varying $N$, while $P = [0, 100]$ for DL-GPR threat (T) alarms. The best-performing classifier is bolded.

<table>
<thead>
<tr>
<th>Supplement Dataset Size (N)</th>
<th>Supplement Dataset Quality (P)</th>
<th>pAUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (No DL Baseline)</td>
<td>[0,100]</td>
<td>0.18</td>
</tr>
<tr>
<td>5</td>
<td>[0,100]</td>
<td>0.29</td>
</tr>
<tr>
<td>10</td>
<td>[0,100]</td>
<td>0.34</td>
</tr>
<tr>
<td>15</td>
<td>[0,100]</td>
<td>0.38</td>
</tr>
<tr>
<td>20</td>
<td>[0,100]</td>
<td>0.39</td>
</tr>
</tbody>
</table>

Table 3.5.2: Performance results when fixing $N\% = 20$, while varying $P$ for DL-GPR threat (T) alarms. The best performing classifier is bolded.

<table>
<thead>
<tr>
<th>Supplement Dataset Size (N%)</th>
<th>Supplement Dataset Quality (P)</th>
<th>pAUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>[0,25]</td>
<td>0.34</td>
</tr>
<tr>
<td>20</td>
<td>[0,50]</td>
<td>0.37</td>
</tr>
<tr>
<td>20</td>
<td>[0,75]</td>
<td>0.30</td>
</tr>
<tr>
<td>20 (Baseline)</td>
<td>[0,100]</td>
<td>0.39</td>
</tr>
<tr>
<td>20</td>
<td>[75, 100]</td>
<td>0.42</td>
</tr>
<tr>
<td>20</td>
<td>[50, 100]</td>
<td>0.41</td>
</tr>
<tr>
<td>20</td>
<td>[25, 100]</td>
<td>0.38</td>
</tr>
</tbody>
</table>

3.5.2. Data Supplementation With Non-Target Data

In this section we present our results investigating supplementation with DL-GPR data collected over *non-threats*. We begin by varying the size of the
supplementation dataset ($N\%$), and sampling the threat alarms uniformly from the total population of DL-GPR prescreener alarms (i.e., $P = [0,100]$). The results of these experiments are presented in Table 3.5.3. The results indicate that any supplementation with DL-GPR non-threat data improves detection performance. However, in contrast to supplementation with threats, performance peaked at $N\% = 10$ and a PAUC=0.32 (compared to PAUC=0.39 for threats) for supplementation with non-threats. This suggests that supplementation with non-threat data is less beneficial than supplementation with threat data. We hypothesize this is because nonthreats are much more variable in their shape and therefore GPR signatures than threats; because of this, nonthreat features may not form clusters in feature space. Therefore, the SVM has fewer points to define support vectors and may be erroneously fitting to what is functionally noise in the data.

Once we established an appropriate value for $N$, we varied the percentile range of prescreener scores, $P$, from which the non-threat supplementary dataset was constructed. The results of these experiments are presented in Table 3.5.4, where $N\% = 10$ for all experiments. Contrary to our hypothesis, the results indicate that varying the percentile range from which the supplementary data is sampled has little impact on detection performance.
Table 3.5.3: Results of varying, $N$, while $P = [0, 100]$ for DL-GPR non-threat (NT) alarms. The best performer is bolded.

<table>
<thead>
<tr>
<th>Supplement Dataset Size (N)</th>
<th>Supplement Dataset Quality (P)</th>
<th>pAUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (No DL Baseline)</td>
<td>[0,100]</td>
<td>0.18</td>
</tr>
<tr>
<td>5</td>
<td>[0,100]</td>
<td>0.29</td>
</tr>
<tr>
<td>10</td>
<td>[0,100]</td>
<td>0.32</td>
</tr>
<tr>
<td>15</td>
<td>[0,100]</td>
<td>0.26</td>
</tr>
<tr>
<td>20</td>
<td>[0,100]</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Table 3.5.4: Results of fixing $N\% = 10$, while varying $P$ for DL-GPR non-threat (NT) alarms. The best performer is bolded.

<table>
<thead>
<tr>
<th>Supplement Dataset Size (N)</th>
<th>Supplement Dataset Quality (P)</th>
<th>pAUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>[0,25]</td>
<td>0.32</td>
</tr>
<tr>
<td>10</td>
<td>[0,50]</td>
<td>0.33</td>
</tr>
<tr>
<td>10</td>
<td>[0,75]</td>
<td>0.32</td>
</tr>
<tr>
<td>10 (Baseline)</td>
<td>[0,100]</td>
<td>0.32</td>
</tr>
<tr>
<td>10</td>
<td>[75, 100]</td>
<td>0.33</td>
</tr>
<tr>
<td>10</td>
<td>[50, 100]</td>
<td>0.32</td>
</tr>
<tr>
<td>10</td>
<td>[25, 100]</td>
<td>0.32</td>
</tr>
</tbody>
</table>
3.5.3. Data Supplementation With Target and Non-Target Data

Based on the results provided in sections 3.5.1 and 3.5.2, we next considered the impact of combining the best supplementary datasets for non-threats and threats, respectively, into a single supplementary dataset for training data augmentation. We compare this approach to the best performing models found in 3.5.1 and 3.5.2, a baseline algorithm which uses no DL-GPR dataset in training, and the chance diagonal for reference. Given that we are only comparing four different algorithms here, we provide the ROCs. The results of this experiment are presented in Figure 3.5.3. Chance performance is calculated by normalizing the chance diagonal (representing 50% accuracy over the entire dataset) by the number of points in a halo that can fit in the total area of the dataset.

The results indicate that combining the best threat and non-threat supplementary datasets into a single supplementary dataset is marginally less effective compared to the best model from section 3.5.1 (which performs the best overall), though this performance difference is unlikely to be statistically significant. What is significant is the performance difference between the best model that uses DL-GPR data and the baseline approach. Indeed, data supplementation with any DL-GPR data is beneficial.
Figure 3.5.3: This figure shows the ROC curves for supplementation. The blue curve represents our baseline which uses no DL data. The orange, yellow, purple, and red-dashed curves represent our best-case results when training a model with supplementary datasets of threat data, nonthreat data, and both types, and the chance diagonal for reference, respectively.

3.6. Summary

In this work we considered the problem of developing algorithms for the automatic detection of buried threats in hand-held Ground Penetrating Radar (HH-GPR) data. Given the similarities between downward-looking GPR (DL-GPR) data and HH-GPR data we explored the possibility of augmenting HH-GPR data with DL-GPR data for training supervised buried threat detection (BTD) algorithms. We assessed the detection performance of a HH-GPR-based BTD algorithm as we vary the amounts, and characteristics of a supplementary DL-GPR dataset added during algorithm training. The results indicate that supplementing HH-GPR data with DL-GPR, either over the
locations of buried threats, or not, improves performance. However, including DL-GPR collected over buried threat locations seems to be sufficient, and adding non-threat data seems to be detrimental when threat data is already added. Furthermore, augmenting with DL-GPR with the strongest threat signals was most beneficial (as judged by a high score assigned by an energy-based detector). This work was published in [23], [24].
4. Neural Style Transfer for Infrared Object Detection

This chapter discusses our work developing ATR algorithms for IR object detection using IR cameras. We aim to develop algorithms that improve the detection performance of deep models for practical applications such as autonomous vehicles or automatic target recognition. To this end our strategy is to leverage large, preexisting RGB datasets and use these in conjunction with IR data to train detection algorithms. The analysis in this chapter also motivates my work in chapter 5.

4.1. Background

Object detection is a computer vision task that has been dominated in recent years by deep neural networks (DNNs) [30]. The rise of deep learning can be partially attributed to the availability of large public benchmark datasets, such as ImageNet [19] and COCO [13], for the training and evaluation of recognition models[1] [30]. Compared to datasets of natural (RGB) imagery, there are relatively few publicly available datasets of IR imagery, and those that are available are much smaller and less diverse than the RGB datasets. This is the problem of data sparsity; in which models do not have enough data to learn effectively.

There are many general strategies to mitigate this data sparsity problem: making greater use of existing data (e.g., data augmentation [12], making more efficient models

---

[1] Other contributing factors include the availability of high-performance parallel computing systems (e.g., GPUs) as well as innovative network architectures and training strategies [4]
that require less data (e.g., ResNet [8]), self-supervised learning [10]), faster data collecting (e.g., synthetic data [12], crowdsourcing data [13]), and transfer learning [69]).

Another strategy to overcome this problem is cross-modal style transfer (CMST) [40]. CMST is a specific case of style transfer, where the goal is to transform imagery collected in one modality (e.g., color) so that it appears as though it were collected using another sensing modality (e.g., infrared). CMST algorithms accomplish this by learning a stylization function between modalities that maps images from one modality to create synthetic images from the other modality. A toy example of RGB–IR CMST is shown in Figure 4.1.1, the synthetic data can then be used to supplement data for training deep models. CMST is used in several applications: autonomous vehicles [70], automatic target recognition [2], visualization [71] [72], and false colorization [73].

![Figure 4.1.1: This figure shows an example of (a) an RGB image, (b) a real IR image and (c) a synthetic IR image produced by a style transfer model. Adapted from [22].](image-url)
CMST can directly leverage the existing large and diverse RGB datasets, making CMST a mechanism by which to efficiently generate synthetic data by using alternative sources of data (in this case, different sensing modalities). This approach has a unique advantage compared to generating data directly from a simulator/generative model or augmentation; that is that CMST only needs to learn a mapping of high-level features (i.e., features that are associated with colors and textures of semantic content in an image) between modalities as the low-level features (i.e., features that are associated with the semantic content and spatial configuration) are already present in the original imagery. Using a generative method to create realistic low-level features from scratch is generally more challenging than learning a mapping between high level features with CMST [12], [40]. Despite that, the mapping function CMST must learn is still complex, nonlinear, and difficult to estimate.

These advantages make CMST a popular method for solving issues associated with data limitations for training. CMST models have previously been employed to solve this RGB–IR task. Algorithms for RGB–IR CMST tend either to be ANST algorithms [15], required paired data from each modality [44], or are not data driven [74]. Previous work demonstrates the potential efficacy of RGB→IR CMST. However, these studies suffer from important limitations: mainly that the authors do not compare their work to many state-of-the-art algorithms [15] [75] or that the authors only evaluate
their proposed work on a limited number of datasets [76] [77]. [75]. Other models RGB→IR CMST models require paired data from each domain such as [78].

To rigorously evaluate the effectiveness of CMST for improving DNN-based IR detectors, we conducted a large-scale comparison of contemporary CMST algorithms. We evaluate several state-of-the-art CMST models (including WCT2 [49], CycleGAN [46], CyCADA [48]) by using a downstream object detection task across multiple public benchmark IR datasets. We use CMST stylized RGB imagery in conjunction with real IR imagery to train a Yolov3 [39] detection algorithm on each dataset and evaluate the resulting detector on sequestered IR test data from each dataset. By using detection accuracy as a metric, we can quantifiably compare CMST models. Surprisingly, data driven CMST algorithms are outperformed by simple grayscale stylization. As we will show in later sections, we conduct an analysis that indicates that existing data driven methods are either too simplistic or too complex, which introduces significant artifacts into stylized imagery. The contributions in this chapter can be summarized as follows:

1. First public comparison of modern CMST approaches for RGB–IR. We present an objective comparison of several state-of-the-art stylization using performance on a downstream detection task as an evaluation metric across multiple benchmark datasets.

2. We perform quantitative and qualitative analyses demonstrating the effectiveness of CMST. We present a comparison of imagery stylized by each algorithm and investigate failure cases of data driven CMST algorithms.
4.2. Related Work

Fundamentally CMST is a special case of image stylization. General image stylization was already reviewed in section 2.3. This section will be a review of work specifically in the field of RGB→IR CMST.

Algorithms for RGB→IR CMST tend either to be ANST algorithms [15], required paired data from each modality [44], or are not data driven [74]. Previous work demonstrates the potential efficacy of RGB→IR CMST. However, these studies suffer from important limitations: mainly that the authors do not compare their work to many state-of-the-art algorithms image stylization algorithms and/or RGB→IR CMST models [15] [75]. In other previous work the authors only evaluate their proposed work on a limited number of datasets [76] [77]. [75]. Other models RGB→IR CMST models require paired data from each domain such as [78] [79]. We assume paired imagery is not available and exclude these models from our review.

In general, there is no comprehensive review (that we are aware of at time of writing) of RGB→IR CMST models that also includes state of the art NST algorithms (both ANST models and PNST models) all evaluated over multiple IR benchmark datasets. Paired CMST models [78] [79], have a systematic advantage over unpaired CMST models and therefore we do not include to them in this review.
4.3. Datasets

In this work we use COCO [13] and multiple public benchmark datasets: FLIR ADAS [22] (FLIR), DSIAC [45], CAMEL [80], and KAIST [81] to train NST algorithms and evaluate detection models. Each dataset was collected with commercial cameras operating in the MWIR band. In this section we discuss modifications made to each dataset (if any) and how we create train/test splits from each dataset. A table of summary statistics of each dataset can be found in Table 4.3.1. Example imagery representative of each of the four IR benchmarks can be seen in Figure 4.3.1.

**FLIR** is a collection of street view IR data developed for autonomous vehicles. The data is partitioned into training and testing divisions by the developers, and we did not make any alterations [22].

**DSIAC** dataset is a collection of MWIR imagery collected by the US Army Night Vision and Electronic Sensors Directorate (NVESD). For our experiments, we subsample DSIAC to only include imagery that contains the people and civilian vehicles (e.g., trucks, SUVs, etc.) as these objects are represented in RGB data. We only include videos with targets which appear at ranges between 500-3500m. Targets of each class are represented at every range for both day and night. Targets moved at a constant velocity in a circular pattern at each range, allowing the targets to be consistently imaged at the same locations all aspect angles. We used random video-based cross validation to create a training dataset and testing dataset which are composed of 80% and 20% of all videos.
respectively. Lastly, we extract every 10th frame from all videos in each set as the imagery to be analyzed, as the camera is static and therefore every video frame is very self-similar to the previous frame. We observed removing every 10th image had every little impact on detection performance.

**CAMEL** is a collection of IR data representing road scenes and pedestrian scenes. We use the train and test splits proposed by [80].

**KAIST** is a collection of street view IR data. We used the same train and test splits used in [37].

**COCO** is a large benchmark dataset of RGB images [13]. It consists thousands of RGB images labeled with ground truth annotations and represents 80 classes. From among the training data, we only use COCO images which contain a class of object that is represented in the IR data and only labels which correspond to those objects (COCO classes 1,2,3, and 18).

Figure 4.3.1: This figure shows an example image from each IR dataset. From left to right: FLIR, DSIAC, CAMEL, and KAIST.
Table 4.3.1: This table lists summary statistics of each benchmark dataset. Note that COCO has been downsampled to only include images and annotations which correspond to the classes of objects that appear in the IR benchmarks. DSIAC has been downsampled by a factor of 10 since many images in that dataset are self-similar.

<table>
<thead>
<tr>
<th>Dataset Properties</th>
<th>Benchmark Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FLIR</td>
</tr>
<tr>
<td># Training images</td>
<td>8.9E3</td>
</tr>
<tr>
<td># Testing images</td>
<td>1.3E3</td>
</tr>
<tr>
<td># Classes</td>
<td>4</td>
</tr>
<tr>
<td># Training Objects</td>
<td>6.7E4</td>
</tr>
<tr>
<td>Average Image Size</td>
<td>512x640</td>
</tr>
<tr>
<td>Average Object Size</td>
<td>38</td>
</tr>
</tbody>
</table>

4.4. Model Implementations

YOLOV3 (YOLO) is the detection algorithm used in this work. The YOLO model is described in 2.2.3 The implementation of the model is by Ultralytics [35]. We trained models using the SGD optimizer [82] with an initial learning rate of 1E-4, and with a step learning schedule that drops the learning rate by a factor of 10 every 10 epochs. The receptive field of the network was 640 pixels. The model weights were pretrained on RGB COCO imagery and fine-tuned on an IR dataset. We use a batch size of eight, where one image in every batch is synthetic IR data. We augment the training data by randomly resizing images, mosaic, random flips, and random crops. Each model is
trained for 40 epochs, except models trained on KAIST, which were trained using the
schedule proposed in [37].

**CycleGAN** is still a widely used model for NST tasks and is frequently used as a
benchmark model in several NST papers [40]. CycleGAN is described in 2.3.1. We
adopted CycleGAN as a representative of the existing methods developed for
adversarial RGB–IR CMST. We used the official implementation described in [46]. We
trained our model using all training images from all IR benchmarks and all training
images from COCO. The parameters of the model were optimized by finding a local
maximum on our downstream task using a forward sequential search.

**CyCADA** is used as a representative example of state-of-the-art adversarial style
transfer models in this work. CyCADA is described in section 2.3.2. We use the official
implementation described in [48]. We trained our model using all training images from
all IR benchmarks and all training images from COCO. The parameters of the model
were optimized by finding a local maximum on our downstream task using a forward
sequential search.

**WCT2** is used as a representative example of PNST models for this study. We
use the official implementation describe in [49]. The WCT2 model is described in
2.3.3. The weights were pretrained on ImageNet, and WCT2 does not require further
training as the transform that matches covariance between the style has a closed form
solution [50]. WCT2 requires image pairs for stylization. Each RGB image is paired with
a random IR image; because COCO is larger than all IR benchmarks combined some IR images are used to stylize multiple COCO images.

**ThermalDet** is included as a non-data-driven model that is more complex than a naïve grayscale stylization. It is a composition of functions from the library of [83]. Specifically, the authors used (in order) five functions: grayscale, invert, gaussian blur, contrast enhancement, and histogram equalization. We preprocess all COCO images using the methods and parameters proposed in [84]. Any parameters that were not specified in the paper, we attempted to find by a greedy sequential forward search.

### 4.5. Neural Style Transfer for Pretraining

There are two ways that CMST can be useful for training: pretraining and fine-tuning. Most (if not all) DNN weights are pretrained on ImageNet, an RGB dataset. Following the idea of [72] presented in the medical community, we hypothesize that if we pretrain on grayscale imagery instead of IR that this will be beneficial to training IR detectors.

#### 4.5.1. Experimental Design

Our experimental design is illustrated in Figure 4.5.1. We train two YOLO models from random initialization on COCO. One version of COCO has been converted to grayscale imagery; the other is the original RGB imagery. We trained each model to convergence on COCO. We then transfer the weights of each model and fine-tune them of each (i.e., one model with grayscale weights one with RGB weights) of our benchmark
datasets. We train a separate YOLO model for each benchmark dataset using the training and testing splits discussed in Section 4.3. Each model is trained for 40 epochs. To measure the performance of our models, we use an IoU=0.5 linking criteria and then measure the Precision and Recall scores of each detector (chapter 2). By the conventions of each dataset, we report mean average precision (mAP) score [22] [45] [80].

Figure 4.5.1: This figure illustrates our experimental design. RGB imagery is stylized by a style transfer model to create synthetic IR images. The synthetic IR imagery is used to pretrain a detection model. The weights of the pretrained model are then fine-tuned on real IR imagery from each of the four benchmark datasets.

4.5.2. Experimental Results

The experimental results are shown in Table 4.5.1. The results indicate that using weights pretrained on grayscale imagery was beneficial to model performance on IR detection datasets. In fact, we observed a performance improvement across every dataset.
We hypothesize two reasons for this increase in performance. First, the model does not inadvertently learn to recognize features in RGB imagery that are not present in IR imagery (e.g., colors) which allows the model to learn features for IR more efficiently. The second is that grayscale stylized images are more similar to IR images, despite that grayscale stylization is simple. These observations motivate our next experiment where we seek to develop more advanced style transfer algorithms and allow RGB data to have a greater effect on the training of models.

Table 4.5.1: This table shows the results of the experiment described in 4.5.1. The first column shows what pretrained weights were used. The middle four columns show the mAP@0.5 score of YOLO models trained and evaluated on each of the benchmark IR datasets. The rightmost column is the average mAP@0.5 score across all datasets.

<table>
<thead>
<tr>
<th>Pretrained Weights</th>
<th>Benchmark Dataset Performance (mAP@0.5)</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FLIR</td>
<td>DSIAC</td>
</tr>
<tr>
<td>RGB (Baseline)</td>
<td>.647</td>
<td>.593</td>
</tr>
<tr>
<td>Gray</td>
<td>.660</td>
<td>.686</td>
</tr>
</tbody>
</table>

4.6. **Neural Style Transfer for Training**

Our experiment in section 4.5 show promising initial results. Pretraining with synthetic IR imagery generated by using an NST algorithm on RGB data was observed to be beneficial, but there are several practical downsides to this method. Chief among them is that it requires the end-to-end training of a neural network from scratch which is computationally expensive and time consuming.
To remedy this, we seek to use CMST stylized RGB imagery directly in training. This undercuts the need for training a model from scratch, allowing for the use of pretrained RGB weights. This has the added benefit of allowing the RGB imagery to have a greater effect on the training of an IR detector. However, this is potentially at the cost of biasing a model toward detecting objects in RGB imagery at the cost of detection performance in the IR domain.

4.6.1. Experimental Design

In our experiments, we systematically create stylized versions of COCO using multiple state-of-the-art and proposed NST algorithms. Each NST model was trained using imagery from all four benchmark IR datasets in aggregate and COCO. We then use the stylized data and real IR data to train a model for each benchmark dataset. To regularize the effect of using stylized data in fine tuning, we construct each minibatch as a mix of stylized and real IR images. A diagram of our experimental design is shown in Figure 4.6.1.

We selected several CMST methods for our experiments. In addition to the data-driven NST models described in 4.4 we also use elementary methods which are not data driven such as: grayscale stylization, inverse grayscale which was observed to be beneficial in [72] [21], and an identity stylization (i.e. ‘do nothing’) representing the most naïve use of RGB imagery for training.
To measure the performance of our models, we use an IoU=0.5 linking criteria and then measure the Precision and Recall scores of each detector (chapter 2). By the conventions of each dataset, we report mean average precision (mAP) score \cite{22} \cite{45}, \cite{80}\cite{85}. We train a separate YOLO model for each benchmark dataset using the training and testing splits discussed in section 4.3. Each model is trained for 40 epochs.

Figure 4.6.1: This figure illustrates mixed batch training. RGB imagery is stylized by a style transfer model to create synthetic IR images. The synthetic IR imagery is used to supplement real IR data in training a detection model. A fixed proportion of each training minibatch is synthetic IR (blue outline) and the remaining images are real IR (orange outline).

4.6.2. Experimental Results

When reporting performance, we report the average mAP score averaged over the last 3 epochs of training. There results of our experiments are shown in Table 4.6.1. There are several observations to elaborate on. The first is that data supplementation is
universally beneficial compared to our baseline result with generic pretraining using
RGB weights, or the best result of our previous experiment where we trained with
gray-scale weights. Even supplementation with unaltered RGB images is beneficial for
our downstream task. The second observation is the dominant performance of models
supplemented with the grayscale stylized imagery. This simple transform yields better
average performance on the downstream tasks than two of the most state-of-the-art NST
algorithms. Third, while grayscale stylization is best on average, it is not ideal to
maximize performance on every dataset. Nor is there an observable trend as to which
model maximizes performance on any given dataset.
Table 4.6.1: This table shows the result of our experiments with CMST methods described in 4.6.2, the winner in each category is bolded. The leftmost column indicates what stylization algorithm was used to create the synthetic IR imagery. The four central columns in each row show the mAP score of Yolov3 models trained with CMST synthetic IR data on each of our benchmark datasets. The rightmost column is the average of the scores on the four datasets. Non data driven methods such as Grayscale outperform data driven models WCT2 and CycleGAN. An unsupervised version of CyCADA is included without using task loss in training.

<table>
<thead>
<tr>
<th>Pretrained Weights</th>
<th>Style Transfer Method</th>
<th>Benchmark Dataset Performance (mAP@0.5)</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FLIR</td>
<td>DSIAC</td>
</tr>
<tr>
<td>RGB (Baseline)</td>
<td>None</td>
<td>.647</td>
<td>.593</td>
</tr>
<tr>
<td>Gray</td>
<td>None</td>
<td>.660</td>
<td>.686</td>
</tr>
<tr>
<td>RGB</td>
<td>RGB</td>
<td>.650</td>
<td>.660</td>
</tr>
<tr>
<td>RGB</td>
<td>Grayscale (inversion)</td>
<td>.676</td>
<td>.660</td>
</tr>
<tr>
<td>RGB</td>
<td>Grayscale</td>
<td>.666</td>
<td>.640</td>
</tr>
<tr>
<td>RGB</td>
<td>ThermalDet</td>
<td>.628</td>
<td>.689</td>
</tr>
<tr>
<td>RGB</td>
<td>WCT2</td>
<td>.642</td>
<td>.660</td>
</tr>
<tr>
<td>RGB</td>
<td>CycleGAN</td>
<td>.600</td>
<td>.669</td>
</tr>
<tr>
<td>RGB</td>
<td>CyCADA</td>
<td>.640</td>
<td>.684</td>
</tr>
</tbody>
</table>

4.6.3. Investigating Failure Cases of NST Models

As mentioned, state-of-the-art models are often either too simplistic, or too powerful and they erroneously alter the content of images. In this section we study the outputs of each model and assess the limitations of each CMST model.

Qualitatively, WCT2 seems to be learning a simple, monotonic transformation for stylization. An implicit assumption of WCT2 (and PNST models in general) is they rely on access to fully segmented labels. While they will function without these labels,
the overall stylization is much less effective. Our IR benchmarks are not fully segmented, nor can we reliably estimate fully segmented labels. Under these conditions, WCT2 is forced to match covariance of global image statistics. This is similar to histogram equalization [86], which is a linear transform.

To confirm this hypothesis, we create several $I_c/I_s$ pairs where we fix the content image but change the style image. An example is shown in Figure 4.6.2. We then regressed the pixel intensities of the corresponding output images $I$ against the pixel intensities of the input we wish to stylize, $I_c$. We observe that for a given $I_c/I_s$ pair, and corresponding output image $I$, that the pixel intensities between the input and output images are always highly linearly correlated. The results of this experiment are shown in Figure 4.6.3. We generalized this result across five different content image and measured the correlation coefficient from fifty $I_c/I_s$ pairs each. We confirmed this to be a general trend and never observed a stylization from WCT2 that couldn’t be approximated by a linear transform, this is shown in Figure 4.6.4. Furthermore, the WCT operation can be modeled as a rotation of features [50], which is a linear operation. Under these conditions, WCT2 doesn’t adequately capture IR style.
Figure 4.6.2: This figure shows example imagery produced by each CMST method we evaluated. Each column of the matrix is five representative images all stylized by an NST algorithm. Gray and WCT2 are both very similar, CycleGAN tends to saturate images, CyCADA introduces artifacts into the imagery.
Figure 4.6.3: Each row of images is a content image $I_c$, style image $I_s$, and output Image ($I$) triple. The final plot in each row is a scatter plot of the pixel intensities of $I$ vs the pixel intensities of $I_c$. A linear fit between these points and the correlation coefficient is shown in red.
Figure 4.6.4: This figure shows the results experiment in Figure 4.6.3 which have been expanded across five different content images, each stylized with fifty different style images (bottom). We then measure the correlation of input and output intensities. We never observed a correlation lower than $\rho = 0.7$, implying the stylization learned by WCT2 is approximating a linear mapping of pixels.

A known problem with CycleGAN is that can remove or change the semantic content of an image [87]. Because our goal is to stylize an object detection training dataset, it is a priority that we preserve the spatial statistics of target imagery. Looking at example images generated by CycleGAN, as shown in Figure 4.6.5. It is clear that CycleGAN is removing content from the images, which is obviously bad. CyCADA stylizations lead to less egregious content destruction, but we observed CyCADA often introduces small artefacts into an image which is also undesirable.
Figure 4.6.5 This figure shows two particularly egregious example of CycleGAN altering the content of an image. Each row is an input image (left) and an output stylized image (right) made with CycleGAN. In both cases, the model attempts to create a stylized output reminiscent of a street view scene from FLIR or Kaist and erroneously removes the surrounding background and even the person in the second row.

4.7. Summary and Conclusions

In this chapter we introduce two benchmark datasets (CAMEL and DSIAC) for consideration by the IR object detection community and rigorously benchmark YOLO models across four datasets. We present a novel means of quantifying the effectiveness of NST algorithms using a downstream task. We performed controlled experiments to
measure the effectiveness of data supplementation with stylized imagery for YOLO algorithms across several IR object detection benchmark datasets. We trained two sets of weights from random initialization on two versions of COCO: RGB and Grayscale. Training with the grayscale weights was found to increase performance on every benchmark IR dataset.

We then tried to increase the effectiveness of our approach by eliminating the reliance on pretrained weights and using more sophisticated state of the art stylization algorithms. Our experiments show that data supplementation with stylized RGB imagery yields higher performance on our benchmark datasets. Disappointingly, state-of-the-art stylization algorithms CycleGAN and WCT2 performed worse than simple grayscale stylization. We then investigated the stylizations learned by each NST models and discussed why these models underperformed.

In the next chapter, we will discuss how our proposed method overcomes these limitations.
5. Meta Learning Style Transfer

This chapter continues the discussion of our work developing ATR algorithms for IR object detection using IR cameras. In this chapter, we build in our contributions in chapter 4 in which conducted analysis showing a comprehensive comparison of several state-of-the-art CMST models and characterized the limitations of these models. We observed that data driven CMST models were inferior, on average, compared to stylization that employed simple hand-crafted image transformations (e.g., grayscale). In this chapter we will address the limitations of existing methods by developing meta learning style transfer (MLST).

Grayscale stylization was the best CMST method we observed, yet when comparing grayscale RGB imagery and real IR imagery there still exists a visible domain gap. We know from physics that an effective CSMT mapping function is more complicated than just a grayscale transform. In principle, data-driven methods can learn much more complex transformations than a grayscale function, however, as we demonstrated in section 4.6.3, such data-driven methods were either (i) too powerful (i.e., they erroneously altered the semantic content of the imagery) or (ii) too simplistic.

The goal of our proposed model is therefore to make the transformation more complex than a simple grayscale operation, but while preventing any stylization model from altering the semantic content in the image. To achieve these goals, we model stylization as a composition of well-defined analytic transformations, as illustrated in
Figure 5.1. The challenge is then to choose the functions that are included in this composition to achieve a realistic stylization. We can leverage prior knowledge about the likely form of the RGB-to-IR transformation (based on physical information and visual assessment) to intelligently choose functions that are most likely to resemble the underlying relationship. At the same time, we limit the dictionary to only include analytic functions with well-defined limitations, to prevent the model from performing operations that are likely to alter the semantic content in the imagery. A similar idea has been used by [84], where the authors handcrafted a stylization function out simple image transformation functions. Specifically, they used (in order) five functions: grayscale, invert, gaussian blur, contrast enhancement, and histogram equalization, but crucially their approach is not data driven.

Figure 5.1: This figure shows an outline of our proposed stylization method, MLST. Stylization is cast as a composition of analytic functions that act on an image in sequence. The functions and their parameters are sampled from a learned probability distribution \( \theta \).
We want to learn what functions and function parameters to use in a data driven way. However, these parameters do not have well defined derivatives [88], and therefore we cannot optimize a model with efficient gradient-descent based methods such as backpropagation [34]. Recent results in the meta learning [89] and automatic augmentation communities [88], [27] propose and evaluate mathematical methods to optimize otherwise non-differentiable functions via backpropagation.

As far as we are aware, no studies in the CMST community have specifically studied the idea of stylization using a dictionary of learned functions. Learning a dictionary of functions is the foundational task of the automatic augmentation community [27], which was reviewed in section 2.4. To infer the best parameters of the compositional model illustrated in Figure 5.1, we minimize the adversarial loss function (section 2.3) from [88].

In [88] this loss was used to perform augmentation, in which the goal of the model was to learn an ideal way to augment RGB imagery, rather than perform CMST. Therefore, the goal was to alter the imagery while still ensuring that the stylized imagery looked like real RGB imagery. This is a different, and potentially simpler, operation than mapping from one modality to the other. It is therefore unclear whether such an adversarial method is sufficient for learning an effective CMST mapping. In this work,
we investigate this question by applying this loss to perform CMST with our proposed MLST model.

5.1. Meta Learning Style Transfer

MLST is trained to learn a composition of analytic functions or ‘stylization policy’ which matches the distributions of stylized RGB images and real IR images using an adversarial critic as illustrated in Figure 5.2.1. The functions in the dictionary available to the model were chosen based on physically and/or visually motivated criteria (i.e., thermal cameras are often lower resolution than RGB cameras of the same aperture size, therefore we include a gaussian blurring operation in the dictionary). MLST is trained in such a way that a policy is only applied to the RGB imagery.

5.2. Meta Learning Style Transfer Methodology

Let a stylization policy be composed of K steps where \( k \in [1,2,\ldots,K] \). At each step, an operation \( O_k^{(n)} \) is applied to an image. Each \( O_k^{(n)} \) is the \( nth \) function in the dictionary. Each \( O_k^{(n)} \) has associated with it a probability of being applied \( p_k^{(n)} \) and a parameter value \( \mu_k^{(n)} \). For each operation it is assumed that \( p^{(n)} \in [0,1] \) and \( \mu^n \in [0,1] \). An image \( X \) is augmented by:

\[
X \leftarrow \begin{cases} 
O^n(X;\mu^{(n)}), & p = p^{(n)} \\
X, & p = 1 - p^{(n)} 
\end{cases}
\]  

(19)
The goal of training is to estimate these parameters that lead to an effective stylization. Both $p_k^n$ and $\mu_k^n$ are learned with gradient descent. Following the method proposed in [55], approximations are made to the gradients of the model to cast error of the critic as differentiable with respect to $p$ and $u$. The probability of a function being applied is sampled from a relaxed Bernoulli distribution:

$$ReBern(b; p, \lambda) = \zeta\left(\frac{1}{\lambda} \left\{ \log\frac{p}{1-p} + \log\frac{u}{1-u} \right\} \right)$$  \hspace{1cm} (20)$$

where $\zeta$ is the sigmoid function that keeps the function in the range $[0,1]$, and $u \in U[0,1]$, $b \in Bern[0,1]$ and $\lambda$ is the ‘temperature’ of the sigmoid function, low $\lambda$ causing $ReBern(b; p, \lambda)$ to behave like a one-hot vector.

Some functions have discrete values of $\mu^n$ (e.g., solarize, invert). In such cases, we use the straight through approximation [90] to estimate the gradient as:

$$\frac{\partial O(X)_{i,j}}{\partial (\mu)} \approx 1$$  \hspace{1cm} (21)$$

where $O_{i,j}$ represent the element of an augmented image by an operation $O$. With these approximations, each function $O^n$ is differentiable with respect to $p^n$ and $\mu^n$. Note that $p_k^n$ and $\mu_k^n$ may change for the same operation at each $k$ step. The parameter $\mu_k^n$ may act as a magnitude or a threshold depending on the operation. The output of a policy can be written as:
\[ X \leftarrow \sum_{k=1}^{K} \sigma(p_k^{(n)})O_k^{(n)}(X; \mu_k^{(n)}) \]

where \( \sigma \) is the softmax function. For a critic, we used a ResNet-18 [8] and replaced the classification layers with a two-layer network. A similar critic is used in [91] [55]. Fundamentally, the training objective of MLST is to match the distribution of stylized RGB imagery and real IR imagery as illustrated in Figure 5.2.2. A classification loss is added to provide the model with more information when training by making use of loose target and source domain labels and to encourage the model to learn a class conditional stylization. Ultimately however, any stylization done to an image is based on a global image transform, which is a weakness of this model. A stylization policy is stochastic, allowing for many possible stylizations of the same image. This is advantageous for our RGB→IR CMST application, as one RGB image can have many IR analogs depending on environmental conditions such as weather or time of day [79]. This stochasticity helps to broaden the distribution of features already present in the training data.
Figure 5.2.1: This figure shows a diagram of our proposed MLST model. RGB imagery is stylized by a learned composition of functions and function parameters (termed ‘stylization policy’) using an adversarial critic for a training signal (dashed red line).
Figure 5.2.2: This figure shows a diagram of the objective of MLST. RGB and IR images form clusters of imagery (red triangles and blue circles respectively) in feature space (gray ellipse). MLST uses the distance between these distributions as a training objective, the model learns a stylization policy that minimizes this distance in feature space between the real IR images and the new distribution of stylized RGB images (purple triangles) while maintaining good classification accuracy on the semantic objects. This is achieved by maximizing the error of an adversarial critic.

5.3. Model Implementation

MLST is based on the ‘AutoAlbument’ implementation of Faster AA proposed by [83]. Our stylization policy has five steps and chooses operations from a pool of eight functions from the library of Albumentations [83]: random brightness, random contrast:
solarize, random gamma transform, random gaussian blur, random gaussian noise, inversion, and identity operations. The definitions of each function are shown in Table 5.3.1.

Table 5.3.1: This table shows the function dictionary available to MLST in our experiments. The left column contains the names of each function. The right contains the definition of each function.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function/parameter description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Brightness</td>
<td>Shift all pixels up/down by a random value $x \in U[-\mu, \mu]$</td>
</tr>
<tr>
<td>Random Contrast</td>
<td>Multiply pixels by a random value $1 + x$, $x \in U[0, \mu]$</td>
</tr>
<tr>
<td>Solarize</td>
<td>Invert all pixels above pixel value $\mu$</td>
</tr>
<tr>
<td>Random Gamma</td>
<td>Exponentiate pixel values by a random value $\frac{x}{100}$, $x \in U[-\mu, \mu]$</td>
</tr>
<tr>
<td>Gaussian Blur</td>
<td>Gaussian blur image with kernel size $k_{size} = (3,3)$ and kernel standard deviation $\sigma = U[0, \mu]$</td>
</tr>
<tr>
<td>Gaussian Noise Variance</td>
<td>Add zero-mean Gaussian noise with variance $\sigma \in U[0, \mu]$</td>
</tr>
<tr>
<td>Invert</td>
<td>Invert image</td>
</tr>
<tr>
<td>Identity</td>
<td>Do nothing</td>
</tr>
</tbody>
</table>

The model was trained using image chips extracted from ground truth boxes and randomly sampled background chips that were sampled from images such that they had no intersection with a ground truth label. We used a learning rate of 1E-3 with the adam optimizer [92] for the policy controller model and critic models. We use a loss weight 100 times higher on the adversarial loss term. We found that our models converged on a
policy around 30 epochs of training. We trained a stylization policy by using all IR datasets in aggregate and COCO.

5.4. Experiments

We used the same experimental design described in 4.5.1 this time using a stylization policy learned by MLST to stylize the RGB imagery. Both MLST and CyCADA make use of source domain labels, which we call supervised NST models. The use of labels gives supervised NST models a fundamental advantage in training compared to unsupervised models (e.g., CycleGAN) which do not make use of additional labels. Because of this systemic advantage, we group our experimental results by supervised and unsupervised NST models (note that both MLST and CyCADA can be trained in an unsupervised way). The results of our experiment on unsupervised NST models are shown in Table 5.4.1. Each row represents a CMST method. The leftmost column indicates what stylization algorithm was used to create the synthetic IR imagery. The four central columns in each row show the mAP score of Yolov3 models trained with CMST synthetic IR data on each of our benchmark datasets. The rightmost column is the average of the scores on the four datasets. MLST performs best on two of the four benchmarks and has the highest average performance of all unsupervised models.

The results of our experiments on supervised models are shown in Table 5.4.2. We added unsupervised MLST (the best unsupervised model on average) to the table for comparison. Unsurprisingly supervised models outperformed their corresponding
unsupervised models. MLST performs best on all benchmarks and has the highest average performance overall, resulting in a 5% increase in mAP score over the best state of the art models such as CyCADA. This is natural because supervised models have access to more information when training. MLST again performed best overall. We also qualitatively compare the stylized imagery produced by data driven algorithms in Figure 5.5.1

Table 5.4.1: This table shows the result of our experiments with unsupervised CMST methods, the winner in each category is bolded. The leftmost column indicates what stylization algorithm was used to create the synthetic IR imagery. The four central columns in each row show the mAP score of detection models trained with synthetic IR data on each of our benchmark datasets. The rightmost column is the average of the scores on the four datasets.

<table>
<thead>
<tr>
<th>Method</th>
<th>FLIR</th>
<th>DSIAC</th>
<th>CAMEL</th>
<th>KAIST</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>None (Baseline)</td>
<td>.647</td>
<td>.593</td>
<td>.582</td>
<td>.427</td>
<td>.562</td>
</tr>
<tr>
<td>Identity</td>
<td>.650</td>
<td>.660</td>
<td>.610</td>
<td>.478</td>
<td>.599</td>
</tr>
<tr>
<td>Grayscale (inversion)</td>
<td>.676</td>
<td>.660</td>
<td>.684</td>
<td>.488</td>
<td>.627</td>
</tr>
<tr>
<td>Grayscale</td>
<td>.666</td>
<td>.640</td>
<td>.693</td>
<td>.478</td>
<td>.619</td>
</tr>
<tr>
<td>ThermalDet</td>
<td>.628</td>
<td>.689</td>
<td>.641</td>
<td>.485</td>
<td>.612</td>
</tr>
<tr>
<td>WCT2</td>
<td>.642</td>
<td>.660</td>
<td>.643</td>
<td>.490</td>
<td>.609</td>
</tr>
<tr>
<td>CycleGAN</td>
<td>.600</td>
<td>.669</td>
<td>.603</td>
<td>.451</td>
<td>.581</td>
</tr>
<tr>
<td>CyCADA</td>
<td>.640</td>
<td>.684</td>
<td>.634</td>
<td>.469</td>
<td>.608</td>
</tr>
<tr>
<td>MLST (Ours)</td>
<td>.629</td>
<td>.716</td>
<td>.749</td>
<td>.459</td>
<td>.638</td>
</tr>
</tbody>
</table>
Table 5.4.2: This table shows the result of our experiments with supervised CMST methods, the winner in each category is bolded. The leftmost column indicates what stylization algorithm was used to create the synthetic IR imagery. The four central columns in each row show the mAP@0.5 score of Yolov3 models trained with CMST synthetic IR data on each of our benchmark datasets. The rightmost column is the average of the scores on the four datasets. Unsupervised MLST (the best unsupervised CMST method) is also included for reference.

<table>
<thead>
<tr>
<th>Method</th>
<th>Supervised?</th>
<th>Benchmark Dataset Performance (mAP@0.5)</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FLIR</td>
<td>DSIAC</td>
</tr>
<tr>
<td>MLST (Ours)</td>
<td>N</td>
<td>.629</td>
<td>.716</td>
</tr>
<tr>
<td>MLST (Ours)</td>
<td>Y</td>
<td>.653</td>
<td>.723</td>
</tr>
<tr>
<td>CyCADA</td>
<td>Y</td>
<td>.647</td>
<td>.686</td>
</tr>
</tbody>
</table>

5.5. Analysis

In this section, we review imagery created by MLST, investigate the stylization policy learned by MLST, and provide evidence the MLST is indeed fulfilling its training objective by stylizing imagery (i.e., as opposed to learning a good augmentation policy). Images stylized by MLST are shown in Figure 5.5.1. We observe that MLST introduces no artifacts or warping into the images (unlike CycleGAN/CyCADA), and the semantic content of each image is preserved. Additionally, the stylization is nonlinear and seems to invert the pixel intensities, increase the contrast of imagery, and the stylized imagery has a subtly blurrier texture.
Figure 5.5.1: This figure shows example imagery produced by each data driven CMST method we evaluated with grayscale imagery included for reference. Each column of the matrix contains five representative images all stylized by an NST algorithm. Gray and WCT2 are both very similar, CycleGAN tends to saturate images, CyCADA introduces artifacts into the imagery. MLST introduces no artifacts and produces a relatively complex nonlinear stylization.
Of course, visually inspecting imagery is speculative. For a more quantitative analysis, we investigated the policy MLST learned. The policy learned by MSLT is shown in Figure 5.5.2. An interesting observation is that the first step of the learned policy is an inverted grayscale transform, which was previously the best stylization observed. To aid in the analysis of the learned policies, it is helpful to compute two summary statistics: the expected number of times each operation is applied to an image:

$$\mathbb{E}[T^{(n)}] = \sum_k p_k^{(n)}$$

and the expected parameter value of an operation:

$$\mathbb{E}[\mu^{(n)}] = \sum_k p_k^{(n)} \mu_k^{(n)}$$

The summary statistics of the learned stylization policy are shown in blue in Figure 5.5.4. We will review the experimental results shown in this figure in the next subsection. For now, the relevant observation is that MLST makes heavy use of specific functions: random contrast, solarize, random gamma, gaussian blur, and inversion. The resulting composition of these functions is nonlinear.
Figure 5.5.2: This figure shows the final policy learned by MLST. Each column represents a step in the policy where an operation is applied to an image. The top row of plots represents the probability of each function in the dictionary being selected at each step. The bottom row represents what function parameters were learned for a given operation in the same step. Note that not every operation has a function parameter (e.g., invert, identity) and their parameters are set to zero. In the limit \( k = 1 \), the stylization learned by MLST is just ‘inversion’ which is equivalent to inverse grayscale stylization, the best non data driven stylization.
5.5.1. Stylization Versus Augmentation?

In this section we investigate whether the performance improvement observed in section 5.4 is truly a consequence of effective stylization, or due to the effects typical of simply augmenting real IR imagery? As discussed in 5.2, stylization is achieved by matching the distribution of features from stylized RGB data with the feature distribution associated with real IR data. Augmentation is different, the goal of augmentation is to expand the existing distribution or IR data by perturbing features with analytic transforms (e.g., color transformations, affine transformations, etc.). Both stylization and augmentation of data creates a richer, more diverse, set of features to train downstream detection models, but is one more beneficial?

To answer this question, we learned an MLST policy for augmentation, and one foe stylization. How each model was trained is illustrated in Figure 5.5.3. We then trained three detection models: one with just an augmentation policy, one with just a stylization policy, and one using both an augmentation and stylization policy following the experimental design in section 5.4.
Figure 5.5.3: This figure shows how MLST is trained for stylization (left) and how MLST is trained for augmentation (right). For stylization only RGB imagery is acted upon by the policy during training. For augmentation IR imagery is acted upon by the policy in training.

The results of our experiment are shown in Table 5.5.1. Both augmentation and stylization yield to better performance on all benchmark datasets. However, MLST for stylization yields a larger mAP score improvement than MLST for augmentation. Additionally, stylization and augmentation are additively beneficial resulting in an even higher mAP.
Table 5.5.1: This table shows difference in performance using MLST for augmentation or for stylization. We see stylization with MLST yields higher performance on downstream tasks compared to traditional augmentation. However, MLST may be used in conjunction with traditional augmentation for even better performance.

<table>
<thead>
<tr>
<th>Method</th>
<th>COCO</th>
<th>Benchmark Dataset Performance (mAP@0.5)</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FLIR</td>
<td>DSIAC</td>
</tr>
<tr>
<td>None</td>
<td>N</td>
<td>.647</td>
<td>.593</td>
</tr>
<tr>
<td>Augmentation Only</td>
<td>N</td>
<td>.632</td>
<td>.657</td>
</tr>
<tr>
<td>Augmentation Only</td>
<td>Y</td>
<td>.646</td>
<td>.664</td>
</tr>
<tr>
<td>Stylization Only</td>
<td>Y</td>
<td>.653</td>
<td>.723</td>
</tr>
<tr>
<td>Stylization + Augmentation</td>
<td>Y</td>
<td>.662</td>
<td>.713</td>
</tr>
</tbody>
</table>

A summary of the different policies learned is shown in Figure 5.5.4. We compare three policies: MLST, MSLT for augmentation, and unsupervised MLST. We see that MLST uses a more diverse number of functions. In particular: random contrast, solarize, random gamma, gaussian blur, and inversion. The policy learned by MLST without task loss is a subset of supervised MLST, using random gamma, gaussian blur, and inversion as well. When used in an augmentation task, or in an unsupervised task, MLST learns a simple inverted grayscale stylization plus gamma correction respectively. One possible explanation for the relatively simple augmentation policy is our choice of function dictionary, which was primarily functions that alter spectral statistics, not spatial statistics. We hypothesize that with a different function dictionary that includes
spatial transformations, an augmentation policy would learn a more diverse set of functions.

Figure 5.5.4: This figure shows the differences in learned policies of MLST when used for stylization (blue), augmentation (orange), and unsupervised MLST (yellow). The left plot shows the expected number of times each operation in the dictionary is applied to an image $E(T(n))$. The right plot shows the average parameter value of each operation $E(\mu(n))$. Note that operations invert and identity have no parameter and their $E(\mu(n))$ is set to zero.

We have shown that the policies learned by MLST for stylization and augmentation are indeed different. To provide further evidence that MLST is meaningfully stylizing RGB imagery we use UMAP [93] to look at the distributions of features of imagery directly by projecting them into a low dimension manifold. We looked at the UMAP projections of features extracted from the 5th convolutional layer of our YOLO model trained on the Kaist dataset. We defined four clusters corresponding to real RGB data, real IR data,
stylized RGB data, and augmented IR data. The UMAP projections of features are shown in Figure 5.5.5. Additional UMAP results on the FLIR dataset are shown in Appendix A.

For both datasets, we observe that (predictably) there is a noticeable difference between RGB and IR imagery. Augmentation expands the distribution of the RGB imagery. MLST produces a distinct cluster of features between the RGB and IR feature clusters, suggesting that MLST has indeed narrowed the domain gap.

Figure 5.5.5: This figure shows the UMAP manifolds of features extracted from a trained YOLO model on Kaist for four classes of image: real IR (blue), RGB (orange), Augmented IR imagery (green), and MLST stylized imagery (blue). Real IR and RGB data form distinct clusters in feature space, indicative of an observable domain gap between imagery. Augmentation expands the distribution of real IR and fills in gaps in that distribution. MLST produces a distinct cluster of imagery between the real IR and RGB images which suggests MLST is closing the domain gap.
5.6. Summary and Conclusions

In this chapter we proposed a novel CMST method, MLST which uses meta
learning to cast CMST as a data driven composition of analytic functions. Building on
the contributions in chapter 4, we conducted controlled experiments to compare our
proposed method with state-of-the-art CMST algorithms using a downstream object
detection task. We observed that MLST performed best of two of the four benchmark
datasets and had the highest average performance overall. We then investigated the
stylization policy learned by MLST and observed that MLST is different (and more
beneficial) than traditional image augmentation. To provide further evidence that MLST
is an effective CMST model we use UMAP to look at the features of RGB imagery, real
IR imagery, stylized RGB imagery, and grayscale RGB imagery. We observed that MLST
produces a distinct cluster of features between the RGB and real IR data, suggesting that
MLST has narrowed the domain gap between the two sensing modalities.
6. Conclusions

The main objective of the work presented in this thesis was to develop novel statistical and algorithmic methods to improve the detection performance of BTD and ATR systems (e.g., increased mAP score or classification accuracy at a given FAR). Our method of achieving this was by using synthetic data created by stylization of existing datasets from related tasks.

In chapter 3, we proposed an algorithm to stylize DL-GPR data to resemble HH-GPR data and then use stylized imagery in conjunction with real HH-GPR data to train HH-GPR detectors. Results show a marginal improvement in pAUC at a fixed FAR.

In chapters 4 and 5, we compared several state-of-the-art CMST methods for RGB→IR style transfer using a downstream detection task of multiple benchmark IR datasets. Results showed that pretraining with grayscale stylized imagery yielded a small improvement across all benchmarks. Further experiments were conducted to study the effectiveness of CMST stylized imagery in training. Results showed that using synthetic imagery in this way resulted in an even greater mAP score improvement compared to pretraining across every benchmark. Yet there was the surprising result that synthetic data created by a grayscale stylization was more beneficial as training data for detectors compared to synthetic data made by complex data driven stylizations learned by state-of-the-art NST models. Further analysis was conducted to show that indeed NST models either are too powerful in that they erroneously alter the semantic
content of images or introduce unrealistic artifacts, or that they are too weak and learn a linear histogram transform.

The method proposed in chapter 5, MLST, addresses the weaknesses of the previously studied models by representing a stylization function as a dictionary of learned of analytic functions. We observed that using MLST stylized imagery in training performed best on two of four benchmarks and was the best algorithm overall of the models we compared to. This was the case comparing to both supervised models and unsupervised models. Analysis was conducted and showed that MLST learns a nonlinear transform that stylizes imagery in a way that is distinct from augmentation while preserving the content of the image.

6.1. Summary of Contributions and Future Work

This document presented several scientific contributions in the field of neural style transfer motivated by the goal of overcoming domain shift by using data driven stylization algorithms to create additional synthetic training data. The main contributions of this work are summarized as follows:

- Leveraging large quantities of downward-looking GPR data to improve landmine ATR algorithms for hand-held GPR. Based upon this work we found that simple preprocessing of downward-looking GPR makes it suitable to train machine learning models that are effective at recognizing threats in hand-held GPR as
well. This leverages many decades of data collection performed on downward-looking GPR to mitigate data collection for hand-held systems.

- **Leveraging massive labeled optical datasets to train DNN-based infrared ATR models.**

  Based upon this work we found that stylizing RGB images makes it suitable to train deep learning models that are effective at recognizing objects in IR imagery. This leverages large, diverse, RGB datasets to mitigate IR data collection.

  - We conducted controlled experiments measuring algorithm performance across multiple benchmark IR detection datasets. Our experiments show that using stylized RGB imagery is beneficial to algorithm performance when used either in the pretraining or fine-tuning of models.

  - We present an objective comparison of several state-of-the-art stylization models using a performance on a downstream ATR task as an evaluation metric.

- **Using meta-machine learning to stylize optical datasets for training DNN-based infrared ATR models.** Based upon this work we found that stylizing RGB images makes it suitable to train deep models that are effective at recognizing objects in IR imagery. Stylizing data in this way yields better performance on ATR tasks than state of the art stylization algorithms.
We developed a novel ‘IR realistic’ stylization to create IR versions of preexisting RGB datasets, like COCO [13]. This method of stylization offers superior performance on downstream detection tasks compared with state-of-the-art stylization algorithms.

We conducted controlled experiments characterizing different augmentation policies learned on multiple different benchmark IR detection datasets.

The new meta learning style transfer method presented in chapter 5 represents a promising new CMST strategy by approximating a stylization function as a learned composition of analytic functions chosen from a dictionary via visual and physically motivated criteria. It was shown that this method produces imagery that is both free of artifacts and preserves the semantic content of the image while also learning a nonlinear stylization function that is able to partially bridge the existing domain gap between RGB and IR imagery more effectively than conventional image augmentation with those same functions. In fact, using synthetic data generated by the proposed method to train a downstream IR detector led to a 15% increase in mAP score compared with conventional transfer learning and a 5% improvement over the best data driven NST models we compared to: doing the best on two of the four benchmark datasets and best on average.
Despite the success of MLST, this method is only valid under certain assumptions and has limitations. MLST assumes loose a priori knowledge of the form of the stylization function. This knowledge can come from physics (e.g., IR cameras are lower resolution than RGB cameras of the same aperture size, therefore RGB→IR stylization should reduce the resolution of the image), or from visual inspection. Based on our remote sensing problem setting, we chose a function dictionary that primarily alters pixel intensities (as opposed to affine transformations or more complex spatial transforms). One avenue of future work is to apply MLST to different sensing modalities such as RGB→Medical (e.g., x-ray, MRI) imagery. Another possible avenue of future work is to expand the dictionary of functions or stylization policies to allow for more complex stylizations.

MLST does have a fundamental limitation: which is that it the stylization policy is a global image transform. This makes the stylization robust and easy to learn but it is not class or context dependent. WCT2, CycleGAN, and CyCADA have the capacity to learn class conditioned stylization if certain assumptions about the data are met, though they are not explicitly incentivized to do so [49] [46] [48].
Appendix A

Figure A.1: This figure shows the UMAP manifolds of features extracted from a trained YOLO model on FLIR for four classes of image: real IR (blue), RGB (orange), Augmented IR imagery (green), and MLST stylized imagery (blue). Real IR and RGB data form distinct clusters in feature space, indicative of an observable domain gap between imagery. Augmentation expands the distribution of real IR and fills in gaps in that distribution. MLST produces a distinct cluster of imagery between the real IR and RGB images which suggests MLST is closing the domain gap.
References


[22] “FLIR Thermal Datasets For Algorithm Training.”

detection using a handheld ground penetrating radar,” no. May 2019, 2020, doi:
10.1117/12.2519949.

[24] E. Stump et al., “How transferable are downward-looking and handheld ground
penetrating radar data? Experiments in the context of buried threat detection,” no.


[27] E. D. Cubuk, B. Zoph, D. Man, V. Vasudevan, and Q. V Le, “AutoAugment:


[29] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese,
“Generalized intersection over union: A metric and a loss for bounding box


[32] D. A. Winkler and T. C. Le, “Performance of Deep and Shallow Neural Networks,
the Universal Approximation Theorem, Activity Cliffs, and QSAR,” Mol. Inform.,

[34] D. E. Rumelhart and G. E. Hintont, “Learning representations by back-


[84] C. Herrmann, M. Ruf, and J. Beyerer, “CNN-based thermal infrared person


Biography

Evan Stump was raised in Southport Connecticut. He attended high school at Fairfield Ludlow Public High School, and then the University of Massachusetts, Amherst where he obtained a Bachelor of Science degree in Physics. He subsequently attended Duke University in Durham North Carolina where he obtained a Master of Science degree in Electrical and Computer Engineering with a focus on signal processing, statistical modeling, and machine learning.