ASSISTED TARGET DETECTION IN AIRBORNE SEARCH AND RESCUE

MARYAM TAHERI-SHIRAZI

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIRMENTS
FOR THE DEGREE OF MASTER OF APPLIED SCIENCE

GRADUATE PROGRAM IN
ELECTRICAL ENGINEERING & COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

October 2020

Abstract

Finding and rescuing people from downed aircraft is challenging in many parts of the world, including Canada. Because the Canadian military still relies on the naked eye to conduct searches, airborne search and rescue could benefit greatly from advanced sensor systems. Partial automation of target detection could alleviate operator workload and potentially improve rescue efforts. One of the obstacles to developing such a system has been the lack of a large, realistic, and ground-truthed search and rescue (SAR) dataset. I used a new dataset for airborne SAR collected in 2014 by the National Research Council Flight Research Laboratory (NRC-FRL) and labeled approximately 40,000 frames, to extract roughly 20,000 negative and 20,000 positive images. Then I tested three ATD methods on this dataset in order to develop more advanced assisted target detection algorithms for thermal infrared (IR) images.

Acknowledgements

I would first like to thank my supervisor, Dr. James Elder, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. Also I would like to acknowledge my colleague at Elder's lab, Dr. Krista Ehinger, I want to thank you for your patient support and for all of the opportunities I was given to further my research. In addition, I would like to thank my parents and my friends for their wise counsel and sympathetic ear. You are always there for me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis I developed an assisted target detection (ATD) algorithm for airborne search and rescue. In this chapter I introduce the motivation behind this project as well as briefly review prior work. I also introduce the methods used in the project, that is deep learning, histogram of oriented gradients, and matched filters.

1.1 Motivation

In Canada, search and rescue (SAR) operations are often performed using manned aircraft. While the Canadian military still relies on the naked eye to conduct searches, advanced sensor systems are poised to transform airborne SAR. Canada's new fleet of SAR aircraft will come equipped with state-of-the-art electro-optical/infrared (EO/IR) stabilized sensor systems that will make it possible to image objects and people several kilometers away.

There are several challenges in applying computer vision to aerial imaging. One is aircraft motion, which can cause motion blur in the image. Also, the small appearance of targets viewed from long distances can result in blurry, low-detail images [1]. Infrared (IR) thermal imagery is widely used because it can function regardless of the time of day. However, even with this technology, it is still difficult for a human operator to reliably detect a target from a moving aircraft when the target may subtend only a few pixels of the display and be partially obscured by weather, terrain, and ground cover [2].

Assisted target detection (ATD) systems can alleviate operator workload and potentially improve rescue efforts. A key limitation in SAR-ATD research has been the lack of a large, realistic, and ground-truthed SAR dataset. In order to asses the capabilities of EO/IR systems, the National Research Council Flight Research Laboratory (NRC-FRL) in collaboration with the Department of National Defense (DND) established a new ground-truthed SAR dataset in

2014, consisting of 22 hours of thermal IR videos from the installed camera on the searching airplane [2]. This motivated us to take advantage of the new NRC-FRL thermal IR SAR dataset to create a new benchmark for ATD systems and evaluate several object detection methods.

1.2 Goals and Contributions

Considerable time was spent cleaning and labeling the NRC-FRL dataset, an effort that can serve us or others in future research. For this purpose, I constructed an algorithm to automatically identify frames containing targets, using log files that accompanied the dataset. I then manually labeled approximately 40,000 frames, to extract roughly 20,000 negative and 20,000 positive images.

Since the ultimate goal of this project is to improve automatic target detection systems, to help with SAR, I tested three ATD methods on this dataset: MF (Matched Filter), HOG + SVM (Histogram of Oriented gradients classified by a support vector machine), and Faster R-CNN (Faster Region-based Convolutional Neural Network, a deep network learning method). Among the three methods, The best performance overall was obtained with Faster R-CNN.

1.3 Prior Work

SAR is a challenging operation performed by human and non-human operators in airplanes, ships and unmanned aerial vehicles (UAVs). The purpose of the SAR is to investigate, find, and rescue targets such as people, airplanes, ships, etc. In recent years, low costs and easier operation have increased the use of UAVs in SAR and surveillance. There are only a few labeled realistic datasets for search in aerial imagery, such as DOTA[3] (a large-scale Dataset for Object Detection in Aerial Images). To get around the limited availability of aerial datasets, Narayanan et al. [4] combined RGB aerial data as well as some synthetic images generated by video game engines to develop an on-board object detection system for hexacopters, improving a pretrained YOLO (You Only Look Once)[5] for the purpose of surveillance.

Some studies used a pre-trained convolutional neural network (CNN). For example, Bejiga et al. [6] used a CNN pre-trained on ImageNet to detect targets (victims in debris) in an avalanche from UAVs, and Stone et al. [7] used a pre-trained CNN to detect explosive hazards in IR images. Despite being trained on different images and for different tasks, these pre-trained neural networks outperformed object detectors based on histogram of oriented gradients (HOG) and its derivatives [7].

IR imaging has many applications in vessel and aircraft search and rescue as well as surveillance and pedestrian detection. IR images can reveal hot or cold targets regardless of illumination.

However, computer vision algorithms used in RGB applications do not necessarily generalize well to IR.

For example, Olmeda et al. [8] found that the wide spectrum of IR sensors and their variability with temperature reduced the reliability of visual features typically used for RGB imagery, such as HOG. Instead, they proposed a novel representation called histogram of oriented phase energy (HOPE), which measures phase alignment across frequency and is more contrast-invariant and reliable for IR imagery. By combining HOPE with a support vector machine (SVM) classifier and a radial basis function kernel, reasonable performance was achieved on a pedestrian detection task.

Herrmann et al. [9] combined a maximally stable extremal region (MSER) proposal generator with a custom-designed CNN for person detection in low-resolution IR imagery, and were able to substantially outperform prior approaches. Elder et al. [10] assessed a potential assisted target detection algorithm for airborne SAR, focusing on nighttime IR sensing. Noting the random variability in data quality due to sensor noise, atmospheric effects, uncertain and changing background terrain, and imperfect target knowledge, they employed a signal detection theory (SDT) approach to maximize expected detection rate.

Another issue for training detectors with IR images of airborne and shipborne surveillance is the limited number of training IR images containing the target of interest and the small size of the object in the image [11]. Wei et al. [12] implemented a method called multiscale patch-based contrast measure to detect small objects in IR guidance systems. They took advantage of the fact that targets were usually brighter than their neighborhood. The main idea was to define a local contrast measure based on patch differences to suppress the background and enhance the targets.

1.4 Matched Filters

A matched filter is a linear filter used to detect a known signal contaminated with stochastic noise. A matched filter is designed to maximize the signal-to-noise ratio [13]. Consider a simple case where the target is a single pixel. The likelihood of the pixel intensity x when projected from the target can be represented by $p(x|T)$ and when projected from the background, can be represented by $p(x|B)$. The likelihood ratio is then:

$$\frac{p(x|T)}{p(x|B)} \tag{1.1}$$

In statistics, this is a likelihood-ratio test. The greater the ratio, the greater the likelihood that the target is present. If I model the conditional distribution of the pixel value as Gaussian, I have:

$$p(x|T) = N(x|\mu_T, \sigma_T) \tag{1.2}$$

$$p(x|B) = N(x|\mu_B, \sigma_B) \tag{1.3}$$

where $N(x|\mu, \sigma)$ is normal distribution with mean μ and standard deviation σ:

$$N(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1.4}$$

The whole image patch X containing either target and/or background consists of s pixels. Assuming the pixel values are conditionally independent, the likelihood of the patch is:

$$p(X|T) = \prod_{i=1}^{s} N(x_i|\mu_T, \sigma_T) \tag{1.5}$$

$$p(X|B) = \prod_{i=1}^{s} N(x_i|\mu_B, \sigma_B) \tag{1.6}$$

where x_i are pixels of the image patch. In practice, the likelihood function is used in logarithmic form:

$$\log(p(X|T)) \propto \sum_{i=1}^{s} -\frac{(x_i - \mu_T)^2}{2\sigma_T^2} \tag{1.7}$$

$$\log(p(X|B)) \propto \sum_{i=1}^{s} -\frac{(x_i - \mu_B)^2}{2\sigma_B^2} \tag{1.8}$$

In the simplest case, $\sigma_B = \sigma_T = \sigma$, and we have:

$$\log \frac{p(X|T)}{p(X|B)} \propto \sum_{i=1}^{s} x_i(\mu_T - \mu_B) \tag{1.9}$$

Equation (1.9) defines the filter as the difference between the target and background expectation (templates). The correlation is calculated by taking the inner product of the patch with the filter.

Note that the matched filter does not account for occlusion, illumination changes, background clutter (that cannot be modeled as an independent and identically distributed Gaussian), scale changes, or rotation. This can limit the performance of the matched filtering approach.

1.5 Histogram of Oriented Gradients

Histogram of oriented gradients (HOG) is a feature descriptor for object detection in computer vision. HOG is a popular baseline in object detection as it is simple, very easy to implement, and performs reasonably well in many tasks. HOG was originally used for pedestrian detection in static images [14]. HOG is based on hand-crafted features. The idea behind it is to describe an object in the image by its distribution of luminance gradient vectors within a patch, computed over uniformly-spaced cells (Figure1.1). Gradient directions are discretized into 9 orientation bins ranging over 0 to 180 degrees; opposite directions are collapsed into the same bin. Within each orientation bin, gradient magnitudes are accumulated.

In each $k \times k$ block of cells, histograms are concatenated into a one-dimensional vector. Usually for RGB images, this vector is normalized to unit length. The final feature is the concatenation of all vectors from each block within a detection window. Finally, I can use a Support Vector Machine (SVM) on top of HOG features for classification. In the next section I review the basics of SVMs.

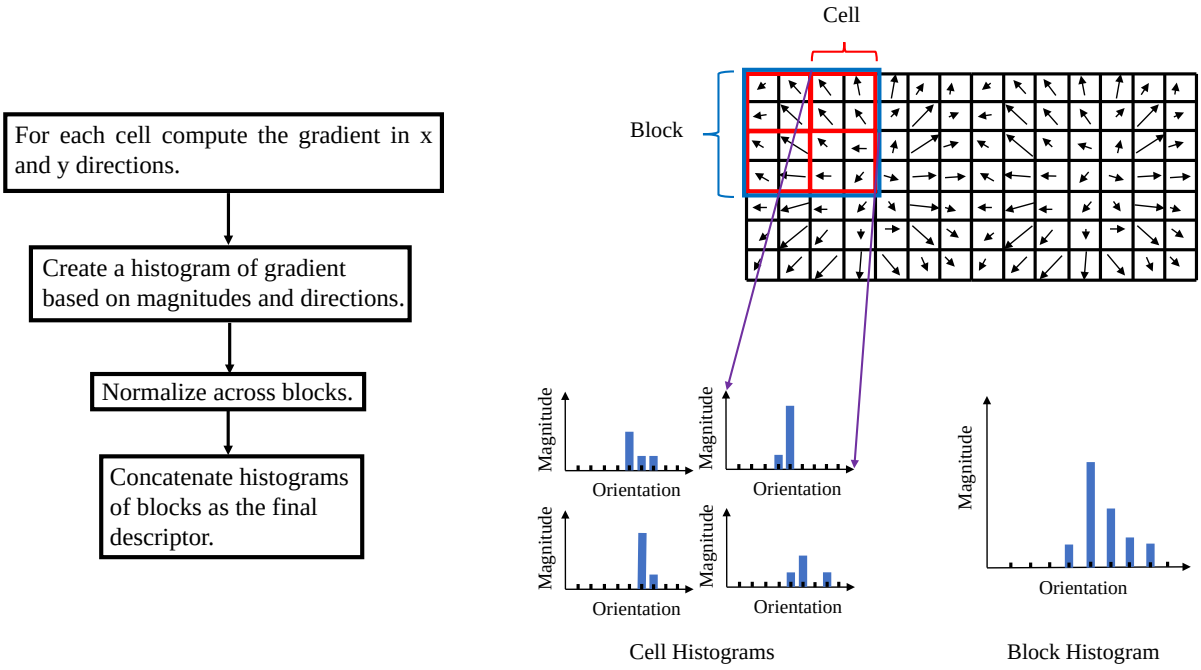

FIGURE 1.1: Histogram of Oriented Gradients

1.5.1 Support Vector Machine

Support vector machines (SVMs) are linear models for classification and regression. An SVM algorithm finds a maximum margin hyperplane decision boundary. This margin separates categories in an N-dimensional space. We choose the hyperplane so that the distance from it to the nearest data point on each side is maximized [15]. Support vectors are the nearest data point to the hyperplane and affect the position and the orientation of the hyperplane. Support vectors define the margin of the classifier as shown in Figure 1.2. Other data points play no part in determining the chosen decision surface.

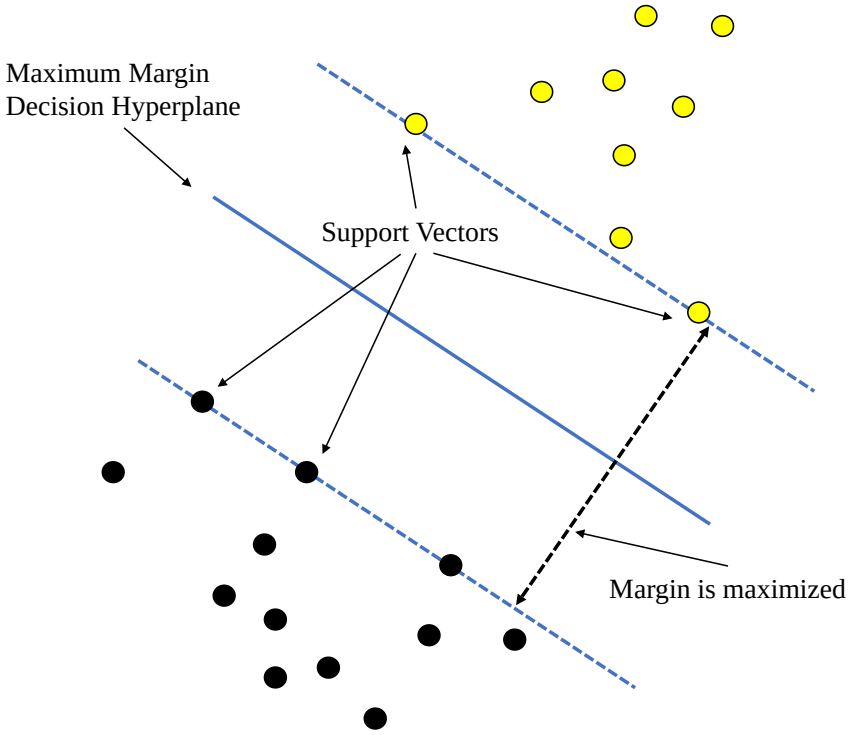

FIGURE 1.2: In an SVM Classifier, support vectors are data points that are closer to the hyperplane and affect the position and the orientation of the hyperplane.

1.5.1.1 Linearly-separable data, binary classification

In binary classification for linearly separable data, the hyperplane, or decision boundary, linearly separates the data. Equation 1.10 shows a linear boundary equation.

$$w^T x + b = 0 \tag{1.10}$$

This linear equation is a multiplication of the weight vector w and data vector x followed by the addition of bias b. Any data vector that yields a positive response is labeled as 1. For example

x_i s.t. $w^T x_i + b > 0$ will have corresponding output of $y_i = 1$. Similarly, any data vector that yields a negative response receives a label of -1. For example, x_i such that $w^T x_i + b < 0$ will have corresponding output of $y_i = -1$.

Using equation 1.10, I define a boundary as well as the nearest data points of either class. I can rescale the data such that anything on or above the line $w^T + b = 1$ is of the class with label 1 and anything on or below the line $w^T + b = -1$ is of the other class with label -1. Let us pick an arbitrary point x_1 on the line $w^T x + b = -1$. Then the closest point on $w^T x + b = 1$ to x_1 is x_2 which satisfies $x_2 = x_1 + \lambda w$. This is because the closest point will always lie on the line orthogonal to the boundaries and passing through x_1, and vector w is orthogonal to both lines. Therefore, $\lambda \|w\|$, the distance between x_1 and x_2, is the distance between the two lines (boundaries). In order to determine λ I start with:

$$w^T x_2 + b = 1 \tag{1.11}$$

substituting $x_2 = x_1 + \lambda w$ I have:

$$\begin{aligned} w^T (x_1 + \lambda w) + b &= 1 \\ w^T x_1 + b + \lambda w^T w &= 1 \end{aligned} \tag{1.12}$$

Since $w^T x_1 + b = -1$ I have:

$$\begin{aligned} -1 + \lambda w^T w &= 1 \\ \lambda w^T w &= 2 \\ \lambda = \frac{2}{w^T w} &= \frac{2}{\|w\|^2} \end{aligned} \tag{1.13}$$

thus, the distance $\lambda \|w\|$ is $\frac{2}{\|w\|^2}\|w\| = \frac{2}{\|w\|} = \frac{2}{\sqrt{w^T w}}$. In order to minimize misclassification, we want this distance to be maximized so that the data points from the two classes lie as far apart as possible. This distance is called the margin. Maximizing the margin corresponds to minimizing $\frac{w^T w}{2}$. Taking into account the linear constraint that points lie on the appropriate side of the decision boundary results in the following quadratic programming problem with linear constraints:

$$\min_{w,b} \frac{w^T w}{2} \tag{1.14}$$

subject to: $y_i(w^T x_i + b) > 1$ ($\forall$ data points x_i).

1.5.1.2 Soft Margin Extension

Consider the case that the data are not perfectly linearly separable. We want to allow some data points to appear on the wrong side of the boundary. To achieve this, we can introduce a slack variable ϵ_i for each data point, such that $\epsilon_i \geq 0$. Then, the quadratic programming problem becomes:

$$\min_{w,b,\epsilon} \frac{w^T w}{2} + C \sum_i \epsilon_i \tag{1.15}$$

which is subject to: $y_i(w^T x_i + b) > 1 - \epsilon_i$ and $\epsilon_i \geqslant 0$ ($\forall$ data points x_i).

I can control the soft margin width with the parameter C. Small C emphasizes the margin while ignoring the outliers and large C may tend to overfit the training data.

1.5.1.3 Higher-Dimensional Spaces

Mapping non-linearly-separable data vectors x_i into a higher-dimensional (even infinite) feature space may make them linearly separable in that space. The formulation remains the same, except that all data points are replaced with $\phi(x_i)$, a function which provides the higher-dimensional mapping.

1.5.1.4 Lagrangian Formulation

In the SVM formulation, for all the data points, the optimal canonical hyperplane should satisfy $y_i(w^T x_i + b) > 1$ and must be close to 1 as possible. We can introduce the Lagrange multipliers to represent this condition. Thus, the new formulation entails the classical quadratic optimization problem with inequality constraints which is solved by the saddle point of the Lagrangian 1.16. The solution corresponding to the original constrained optimization is always a saddle point of the Lagrangian function. A saddle point is a point on the surface of the graph of a function where the derivatives in orthogonal directions are all zero.

$$L(w, b, \alpha) = \frac{w^T w}{2} - \sum_i \alpha_i [y_i(w^T \phi(x_i) + b) - 1] \tag{1.16}$$

L has to be minimized with respect to (w, b) and maximized with respect to α_i, where the α_i are Lagrangian multipliers [16]. In order to prevent α_i from going to ∞, I can impose constraints on the Lagrangian multipliers to satisfy this condition: $0 \leqslant \alpha_i \leqslant C$. Thus, the problem becomes:

$$\min_{w,b}(\frac{w^T w}{2} - \sum_i \max_{\alpha_i} \alpha_i [y_i(w^T \phi(x_i) + b) - 1]) \tag{1.17}$$

Interchanging the max and min, I obtain:

$$\max_{\alpha_i} \min_{w,b}(\frac{w^T w}{2} - \sum_i \alpha_i [y_i(w^T \phi(x_i) + b) - 1]) \tag{1.18}$$

To solve this optimization problem, I set $\frac{\partial L}{\partial w} = 0$ which results in the optimal solution $\sum_i \alpha_i y_i \phi(x_i)$. Setting $\frac{\partial L}{\partial b} = 0$ yields the constraint $\sum_i \alpha_i y_i = 0$. Substituting and simplifying, I get:

$$\min_{w,b} L(w, b; \alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) \tag{1.19}$$

Therefore the problem becomes:

$$\max_{\alpha_i}(\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j)) \tag{1.20}$$

subject to: $\sum_i \alpha_i y_i = 0$ and $0 \leqslant \alpha \leqslant C$

1.5.1.5 Kernel Trick

Computing the inner product $\phi(x_i)^T \phi(x_j)$ directly is expensive if the feature space is high-dimensional, and impossible if it is infinite-dimensional. The kernel trick is a method for efficiently determining this inner product without having to directly compute the feature vectors $\phi(x_i)$. The kernel trick merges the mapping of input vectors to a higher-dimensional space and the forming of an inner product in that space into a single kernel function k. This allows me to rewrite Equation 1.20 as:

$$\max_{\alpha_i}(\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)) \tag{1.21}$$

The radial basis function kernel, or RBF kernel, is a popular kernel function used in kernelized learning algorithms:

$$k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2}) \tag{1.22}$$

where $\|x_i - x_j\|^2$ is squared Euclidean distance between the two vectors. σ is a free parameter. SVMs are memory efficient but do not provide probabilistic measures of uncertainty.

1.6 Deep Learning

Deep learning is a sub-field of machine learning that is highly popular due to its impressive performance in artificial intelligence domains, such as computer vision, natural language processing, and robotics. Deep learning uses a hierarchical neural network architecture to learn high-level features [17]. Convolutional neural networks (CNNs) in particular are commonly used in computer vision applications. One of the most important algorithms in deep learning is back-propagation as it computes an efficient gradient of the loss function with respect to the weights in a supervised neural network. It uses the chain rule for calculating derivatives. The gradient is calculated sequentially from the output layer back to the input layer. Since the gradient at any given layer can be expressed as a simple function of the gradient at the following layer, this reduces the computation time. However, since the loss function is typically non-convex, standard gradient descent can easily get trapped in local optima. Stochastic gradient descent calculates the gradient based upon a small random sample of the training dataset, rather than the entire training dataset. This gives the algorithm the opportunity to jump between local wells in the objective function increasing the probability that a near-optimal solution will be found, and also greatly reduces the computation required for a single weight update.

Generally speaking, a deep network consists of a hierarchical architecture with many layers, each of which consists of alternating linear and non-linear processing steps that may include convolutional layers, pooling layers, and fully-connected layers. Each layer has a different role in the architecture of the network.

Convolutional Layers:

Convolutional layers serve as feature extractors, and they learn feature representations of the input image [18]. They consist of sets of filters or kernels. Through the forward pass in the neural network, each filter in a convolutional layer is convolved with the input image and computes a new feature map [19]. Convolved results are sent through a nonlinear activation function. Different feature maps within the same convolutional layer have different weights so that several features can be extracted at each location [19, 20]. This transformation can be represented as:

$$Y_k = f(W_k * x) \tag{1.23}$$

where the input image is denoted by x, and the convolutional filter of the k_{th} feature map is denoted by W_k. The asterisk refers to the 2D convolutional operator, which is used to calculate the inner product of the filter model at each location of the input image. The $f(.)$ represents the nonlinear activation function.

Nonlinear activation functions allow for the extraction of nonlinear features. Traditionally, the sigmoid and hyperbolic tangent functions were used; recently, rectified linear units [21] have become popular [19].

Pooling Layers:

The purpose of the pooling layer is to reduce the spatial resolution of the feature maps and achieve spatial invariance to input distortions and translations. Initially, it was common practice to use average pooling aggregation layers to propagate the average of all the input values of a small neighborhood of an image to the next layer. However, in more recent models, max pooling aggregation layers propagate the maximum value within a receptive field to the next layer. Formally, max pooling selects the largest element within each receptive field:

$$Y_{kij} = \max_{(p,q \in R_{ij})} x_{kpq} \tag{1.24}$$

where x_{kpq} denotes the activation at location (p, q) within the pooling region or receptive field R_{ij}, at the position (i, j) [22].

Fully Connected Layers:

Several convolutional and pooling layers are usually stacked on top of each other to extract more abstract feature representations when going deeper through the network. The fully connected layers that follow these layers interpret these feature representations and implement a decision stage. For classification problems, it is standard to use the softmax operator on top of a deep convolutional neural network [23, 27–30].

One CNN architecture is the region-based Convolutional Neural Network (R-CNN) [24]. R-CNNs can be trained to find objects as well as their locations. This is relevant to my project (assisted target detection) as I wish to locate the downed aircraft in the image. Sliding window search is a conventional method to scan over the image in order to detect the object in the image and define the object location with respect to the location of the sliding window. In R-CNN, the algorithm creates multiple object proposals as an alternative to a sliding window search. R-CNNs feed region proposals at the pixel level to the CNNs and extract features for

each region to find and locate objects in the image. Classification is performed using an SVM on the features from each region. Because R-CNNs compute the feature map separately for each region, they are slow.

The new generations of R-CNN are Fast R-CNN and Faster R-CNN. In Fast R-CNN a single network computes the whole feature map first, classifies each proposed region of the feature map, and finally defines the bounding boxes [25]. Faster R-CNN replaces the selective search used in Fast R-CNN with the same CNN used for feature map extraction. Thus one single network creates the feature maps, proposes regions, classifies, and defines the bounding boxes [26].

1.6.0.1 R-CNN

The goal of object detection is to identify the class of the object as well as the bounding box size and location. Conventionally, for each image, there is a sliding window to search every position within the image as below, see Figure 1.3. Objects can have different aspect ratios and sizes depending on object size and distance from camera. If I use deep learning for image classification at each location, it will be a slow process.

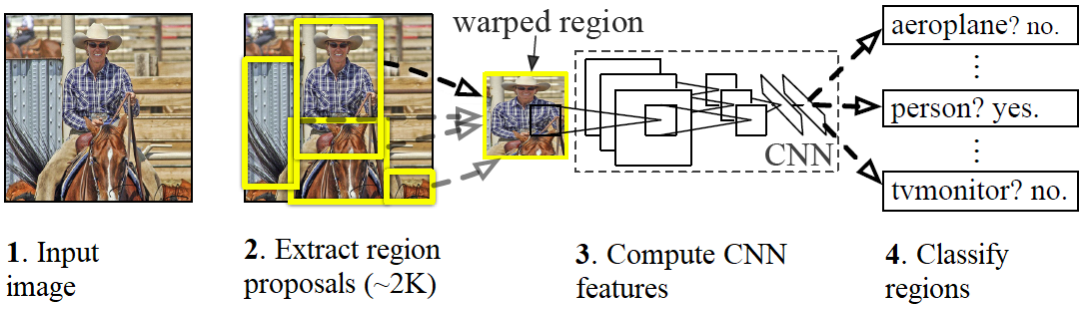

FIGURE 1.3: Region-Based Convolutional Neural Network Structure. R-CNNs feed region proposals to CNNs and extract features for each region to find and locate objects in the image. Classification is performed using an SVM on the features from each region [24].

R-CNN uses selective search to generate proposals (bounding boxes) for image classification [31], [32]. In selective search, color similarities, texture similarities, region size, and region filling are used as non-object-based segmentation, see Figure 1.4. Therefore, there will be many small segmented areas. Then using a bottom-up approach, small segmented areas are merged together to form larger segmented areas, generating roughly 2k region proposals in the image.

The image in each proposed bounding box is fed to a CNN. Since windows have various sizes, all bounding boxes are warped to a certain size. In R-CNN, AlexNet, as shown in Figure 1.5, is used to extract the CNN features. For each proposal, a feature vector is computed by forward propagating the mean-subtracted RGB image patch through five convolutional layers and two

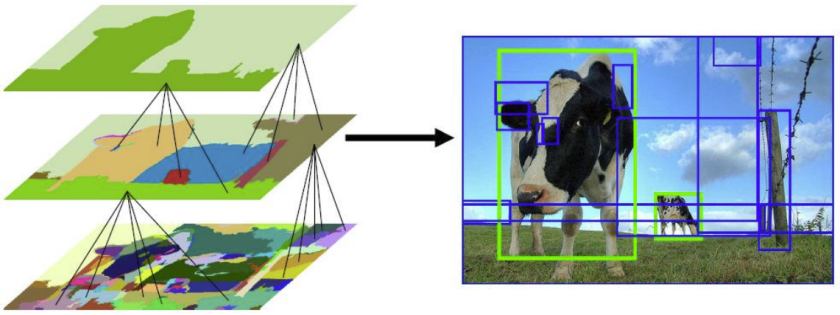

FIGURE 1.4: In selective search, color similarities, texture similarities, region size, and region filling are used as non-object-based segmentation [24].

fully connected layers. Then the feature vector is scored by an SVM trained for each class. A non-maximum suppression (NMS) process eliminates weaker detections whose bounding boxes overlap with stronger detections. Finally the bounding box locations, sizes and aspect ratios are refined by regression.

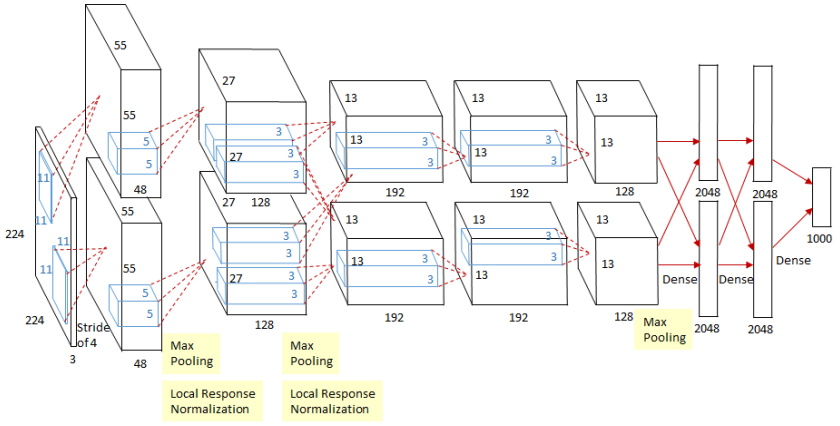

FIGURE 1.5: AlexNet Neural Network Structure. In R-CNN AlexNet is used to extract the CNN features, based on figure in [23].

In R-CNN, softmax classifiers are trained first and then R-CNN uses the feature vectors to train the bounding box regressor. R-CNN is not an end-to-end training algorithm since it has a multi-stage pipeline as shown in Figure 1.3. It is an expensive algorithm in space as well as in time.

1.6.0.2 Fast R-CNN

The overall approach of Fast R-CNN is similar to R-CNN but instead of feeding the region proposal to the CNN, it feeds the input image to the CNN to generate the convolutional feature map. Then from this map, it can identify the region proposals and warp them into squares using a region of interest (ROI) pooling layer. The pooling size of a max-pooling layer in an

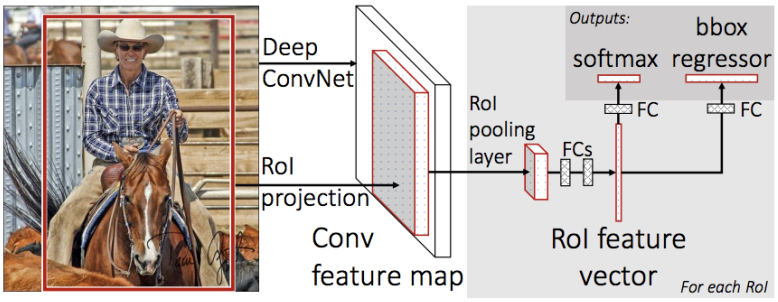

FIGURE 1.6: Fast R-CNN Structure. The input image is fed to the CNN to generate the convolutional feature map. Figure taken from [25].

ROI pooling layer is dependent on the input size to ensure that the output size is always the same. This layer is essential because the fully-connected layer always expects the same input size.

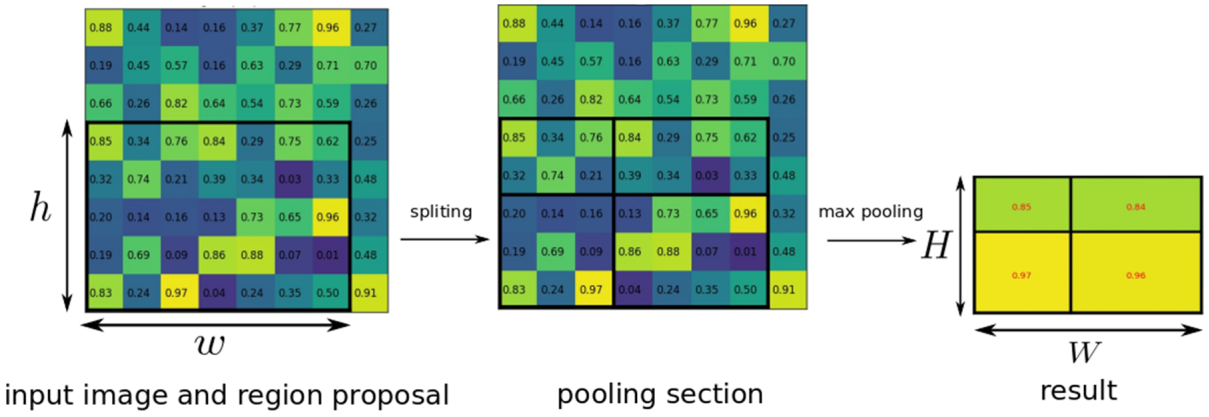

FIGURE 1.7: Region of interest pooling layer (RoI). This layer is essential since it assures that all the proposals have the same size. Figure taken from [25]

Fast R-CNN is an end-to-end learning algorithm. Since it learns the features of objects as well as the associated bounding boxes and their sizes, the loss function of Fast R-CNN is a multi-task loss function consisting of the classification loss and the bounding box loss.

1.6.0.3 Faster R-CNN

In Faster R-CNN, the main idea is to use deep convolutional layers to infer region proposals. Faster R-CNN consists of two modules as shown in Figure 1.8:

1. Region Proposal Network:

The Region Proposal Network (RPN) gives a set of rectangles (anchors) based on the deep convolutional layers. In the default configuration of Faster R-CNN, there are nine anchors at each location, with aspect ratio of 1:1, 1:2, and 2:1.

2. Region of interest pooling layer:

   The region of interest pooling layer (RoI) classifies the object in each proposal and refines the proposal location accordingly.

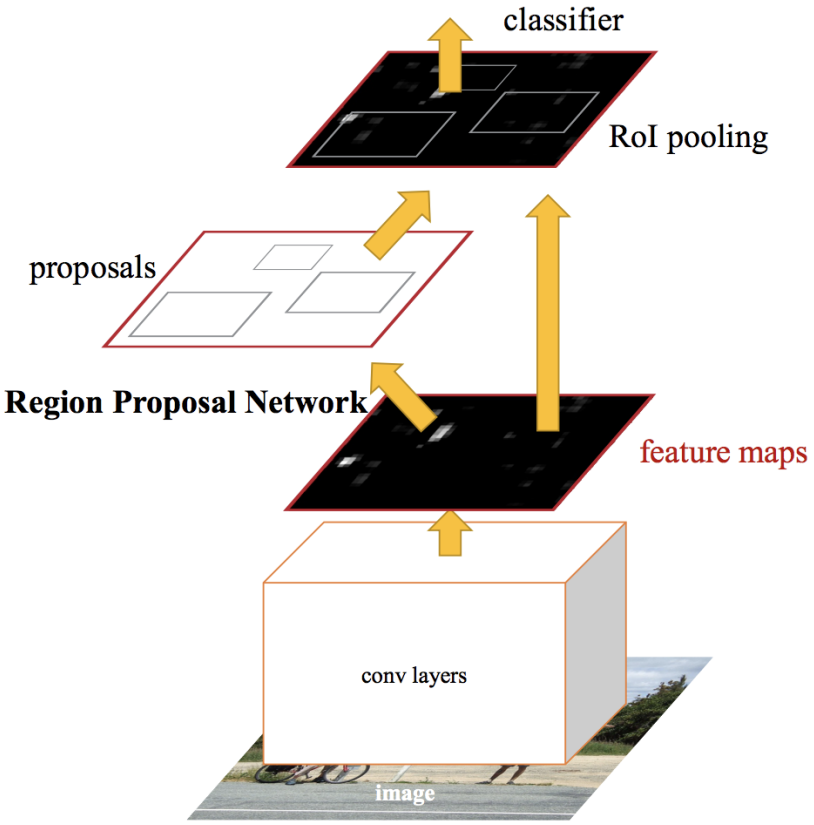

FIGURE 1.8: Faster R-CNN consists of an RPN and a region and an RoI layer. Figure taken from [26].

There are steps of Faster R-CNN algorithm:

1. We need to choose a convolutional neural network structure for the desired object detection.
2. At this point, the algorithm defines the feature maps from the deep convolutional layers.
3. We train a Region Proposal Network (RPN) that detects the object in the image and defines the bounding box location in the image.
4. The algorithm feeds the proposals to an ROI pooling layer such as Fast R-CNN.

5. Finally, the algorithm reshapes all proposals to a fixed window size and sends them to a fully-connected layer to predict a class label.

For every point in the output feature map, the network has to learn whether an object is present in the input image at the object's corresponding location and estimate its size. This is done by placing a set of anchors (boxes) on the input image for each location on the output feature map from the backbone network. These anchors indicate possible objects in various sizes and aspect ratios at each location. In the default configuration of Faster R-CNN, there are nine anchors with different positions in the image. For example, for an image of size 600×800 pixels, I can define three scales, such as 128×128, 256×256, and 512×512. The nine anchors will have the height and width ratio of 1:1, 1:2, and 2:1 with respect to each of the scales.

As the network moves through each pixel in the output feature map, it has to check whether these k corresponding anchors spanning the input image actually contain objects, and then refine the anchors' coordinates to define the bounding box in the image.

At the training stage, all of the anchors that cross the image boundaries are ignored so that they do not contribute to the loss. An anchor is considered to be a positive sample if it satisfies either of the two following conditions: first, the anchor has the highest intersection over union (IoU) with the ground truth or the anchor overlaps by more than 70% with any ground truth box. The same ground truth box can cause multiple anchors to be assigned positive labels. The anchor is labeled negative if its IoU with all ground truth boxes is less than 0.3. Each mini batch for training the RPN comes from a single image. An equal number of positive and negative samples are randomly selected for the training. The training loss for the RPN is:

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i*) \tag{1.25}$$

where i is the index of the anchor in the mini-batch, t_i is the predicted coordinate and t_i^* is the ground truth coordinate. The $L_{cls}(p_i, p_i^*)$ is the classification loss (cross-entropy). The output score from the classification branch for anchor i is p_i, and p_i^* is the ground truth label, which can be either zero or one. The regression loss $L_{re}(t_i, t_i^*)$ is activated only if the anchor actually contains an object. The variable t_i is the output prediction of the regression layer.

At test time, all boxes are sorted according to their class scores. Then non-maximum suppression (which selects the window with the higher score from a given number of overlapping windows) is applied with a threshold of 0.7. Starting from the highest-scoring box, all of the bounding boxes having an IoU greater than 0.7 with previously visited bounding box having been discarded.

In R-CNN, for each proposed region, a CNN is employed, making it computationally expensive. This is solved in Fast R-CNN by generating the convolutional feature map directly from the

input image. The selective search component of Fast R-CNN is a computation bottle neck that has been replaced with a Region Proposal Network in Faster R-CNN, making the detection and localization even faster.

1.7 Conclusion

In this chapter, I reviewed the basics of the methods used in this project which are matched filters (MF), histogram of oriented gradients (HOG), and Faster R-CNN, a deep learning method. In MF, I search for image patches similar, in a least-square sense, to a linear template learned from training data. In HOG, detection is based on the distribution of the local gradient vectors. These methods will be baselines for comparison to the more sophisticated R-CNN detector in my project.

Chapter 2

Dataset

In order to develop an assisted target detection system, this project uses the dataset developed by the National Research Council Flight Research Laboratory (NRC-FRL). NRC-FRL conducted a series of flights to assess the capabilities of the sensor operators using an EO/IR system versus traditional night goggles for search and rescue. The flight experiment, using an NRC Twin Otter aircraft, was conducted between September 8 and October 3, 2014. Over a period of 2-3 days, 90 targets were placed by four teams of NRC and military personnel and Civil Air Association volunteers in a 200 km^2 area as shown in Figure 2.1.

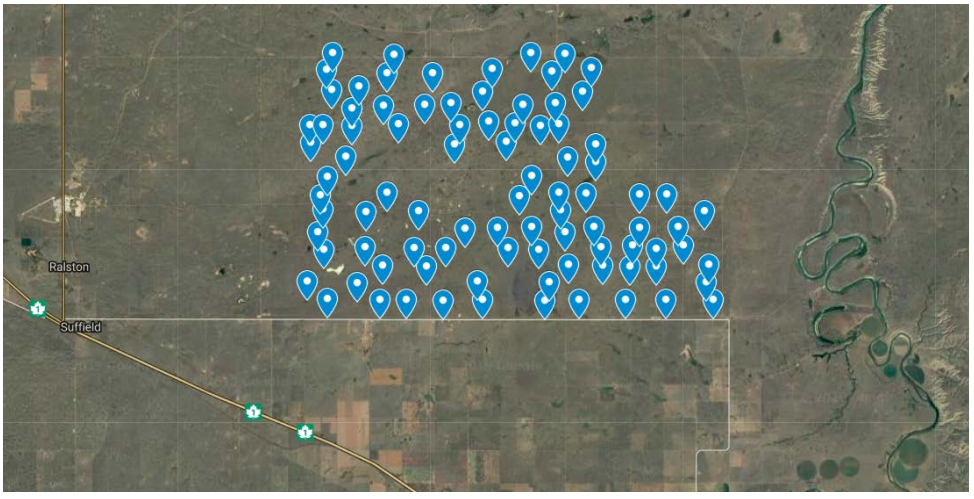

FIGURE 2.1: Defense Research and Development Canada-Suffield, Alberta: Targets shown by blue pins in a 200 km^2 area.

The Twin Otter flew multiple parallel tracks over the designated test area, during which two Department of National Defense (DND) EO/IR operators were tasked with finding targets in the test area. One sensor operator used an EO/IR system to detect and discriminate between potential targets and non-targets and the other sensor operator detected targets with night vision goggles. The two sensor operators switched detection roles on alternate nights.

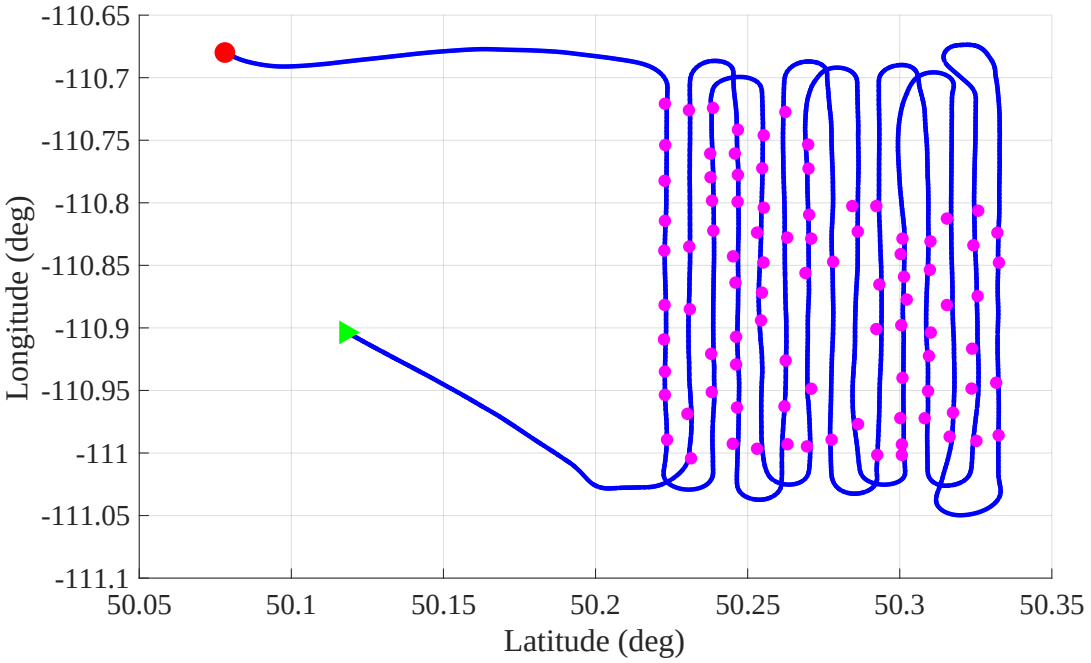

FIGURE 2.2: Flight lines are shown with blue color. The green triangle is the start point, and the red circle is the end point. The magenta circles are the locations of the targets placed aligned with the flight lines.

The targets consisted of pieces of lightweight, flexible, reflective, aluminized polyethylene, arranged in a typical aircraft shape. The test flights were conducted at night (after local astronomical twilight), each approximately three to four hours in duration. The targets were placed along 15 flight lines, each 11 km in length. The 15 lines combined resulted in a 315 km long track which took approximately 2 hours to travel. Each line was traversed once in each direction. Figure 2.2 shows a sample of the tracking flight lines. A total of 12 flights were conducted over a period of three weeks. Some of the targets changed in appearance over time due to damage from weather or animals.

The NRC Twin Otter was flown over the test area at 1500 feet above ground level at 100 knots. The sensor operators were instructed to direct the EO/IR sensor forward along the aircraft track. The sensor elevation was set at 30 degrees below the horizon during the search, which put the center of the sensor field of view approximately 3000 feet in front of the aircraft.

Over 22 hours of data collection was completed during the flight test period. Data was saved in the format of a transform stream file. Videos were paired with log files containing flight information such as location of the airplane (in longitude and latitude), heading, pitch and roll angles, field of view of the camera, and the location of the corners of the field of view in longitude and latitude (ground point back projections).

Generally, the target material (polyethylene plates) was highly visible to the EO/IR systems. Targets had a consistent shape and sat on relatively flat terrain with few trees. However, the

probability of detection (POD) by human operators for these targets was not perfect, with an average performance of 81.8 ± 5.7 %. The targets were typically cooler than the background grassland but there were some targets that were placed in fairly wet (i.e. cool) areas. These cooler targets were usually not seen with the EO/IR sensor.

Figure 2.3 shows samples from the dataset. The deployed IR sensors had a resolution of 640×480 pixels and a scanning field of view of 18×14 deg, operating at 30 frames per second. The human operator could zoom in on the targets during the operation. Targets subtended about 20×20 pixels of each image when the lens was zoomed out (wide angle), as it would be in scan mode. For the most part, the polarity of the IR sensor was set to white-hot, meaning the search objects were typically darker (i.e., colder) than the background terrain.

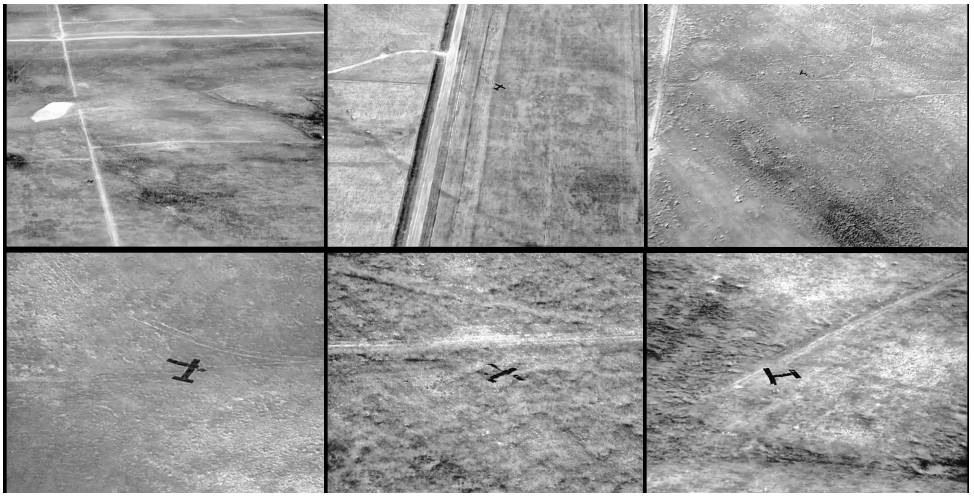

FIGURE 2.3: Dataset samples. Top row: zoomed-out images. Bottom row: zoomed-in images.

2.1 Data Preparation

The goal of this project is to utilize deep neural networks to create a new generation of assisted target detection algorithms with higher performance than previous algorithms. To train a deep neural network, I need many ground-truthed images. The new dataset from NRC-FRL consists of over 22 hours of videos with a frame rate of 30 fps, providing us with more than 2 million frames. To avoid searching for the targets manually, I used log files of the Twin Otter (including airplane position, location, and target GPS coordinates) to estimate and identify video frames containing targets. These log files were recorded in different formats. The main log files are CSV files and as shown in Table 2.1, they contain various information about the search flight.

I used the main log file information to associate the target's location on the ground to the video frames (Figure 2.5). I can recreate the ground plane field of view (FOV), as shown in Figure 2.4, using the four corners of the back-projected field of view of the installed sensor in

UNIX Time Stamp (microsecond)	Mission ID
Platform Tail Number	Platform Heading Angle (deg)
Platform Pitch Angle (deg)	Platform Roll Angle (deg)
Platform Designation	Sensor Latitude (deg)
Sensor Longitude (deg)	Sensor True Altitude (meters)
Sensor Horizontal Field of View (deg)	Sensor Vertical Field of View (deg)
Sensor Relative Azimuth Angle (deg)	Sensor Relative Elevation Angle (deg)
Sensor Relative Roll Angle (deg)	Slant Range (meters)
Frame Center latitude (deg)	Frame Center longitude (deg)
Frame Center Elevation (meters)	Offset Corner Latitude Point 1 (deg)
Offset Corner Longitude Point 1 (deg)	Offset Corner Latitude Point 2 (deg)
Offset Corner Longitude Point 2 (deg)	Offset Corner Latitude Point 3 (deg)
Offset Corner Longitude Point 3 (deg)	Offset Corner Latitude Point 4 (deg)
Offset Corner Longitude Point 4 (deg)	Platform Call Sign
UAS LDS Version Number	Event Start Time - UTC

TABLE 2.1: Videos are paired with log files containing flight information such as location of the airplane (in longitude and latitude), heading, pitch and roll angles, field of view of the camera, and corners of the field of view (ground point back projections).

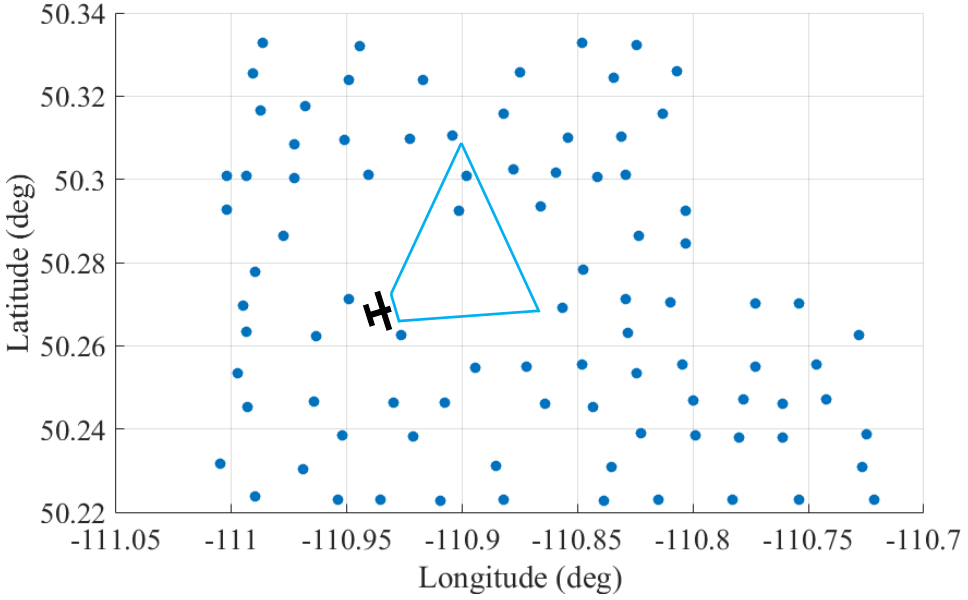

FIGURE 2.4: Typical airborne search geometry. The polygon represents the sensor field of view, projected on the ground plane. The asymmetry is due to the roll of the airplane. Blue points represent target locations.

the log files. Whenever a target appears in the FOV polygon, I can correlate information such as location of the target recorded by NRC-FRL staff with the video frame that contains this target and extract the frames with at least one target as positive samples of the dataset. I used a homography to project the location of the target in the polygon to the image coordinates, as explained in section 2.1.1 and at the end I extracted about 20,000 positive frames for the dataset. In section 2.2 I discuss the issues with extracting frames with targets.

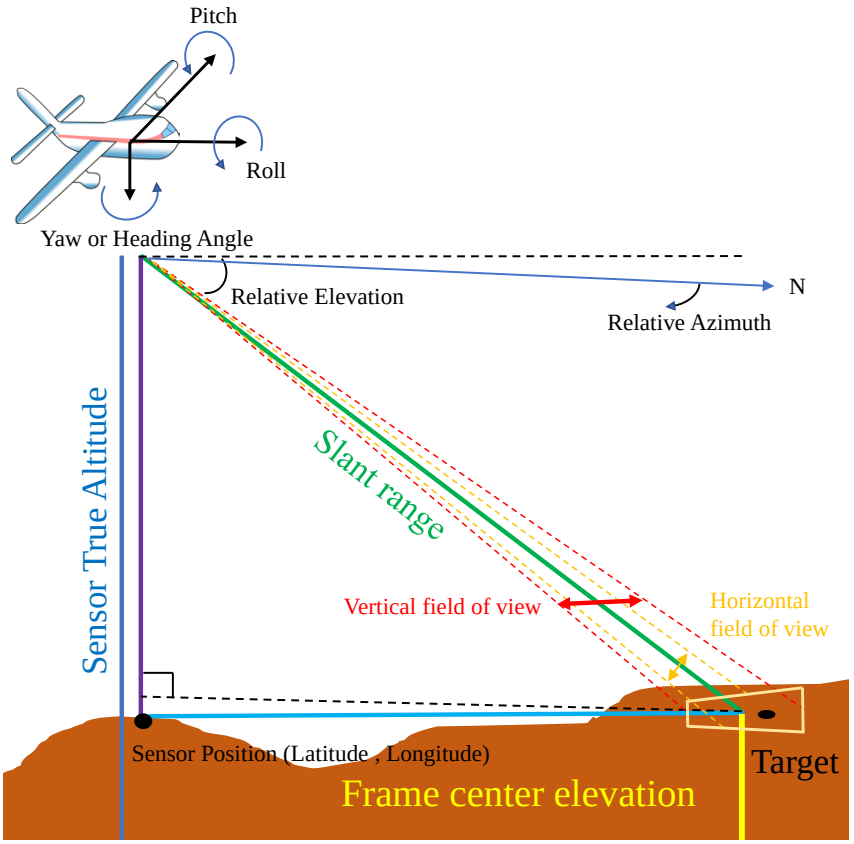

FIGURE 2.5: Key log file variable.

2014-09-11	There is a 15-second gap after each 15 seconds of recorded data.
2014-09-16	No information was recorded in CSV files.
2014-09-18	No CSV files come with the videos.
2014-09-20	Very few videos and CSV files were provided.
2014-09-23	Data storage convention not followed.
2014-09-29	No information was recorded in CSV files.

TABLE 2.2: Days with recording failures. I have not considered these days in our analysis.

As shown in Table 2.2, 6 days of the 15 days data collection were excluded due to problems with the data files, leaving 9 days for our dataset. Using the information in the log files, I can estimate the flight path for each day.

As shown in Figure 2.6, I simulated the flight path for each day. The green triangle shows where the human operator started recording data, and the red circle shows where data recording ended. The locations of the airplane when the human operator took snapshots of the target on the ground are shown with black triangles and the targets on the ground are shown with magenta circles.

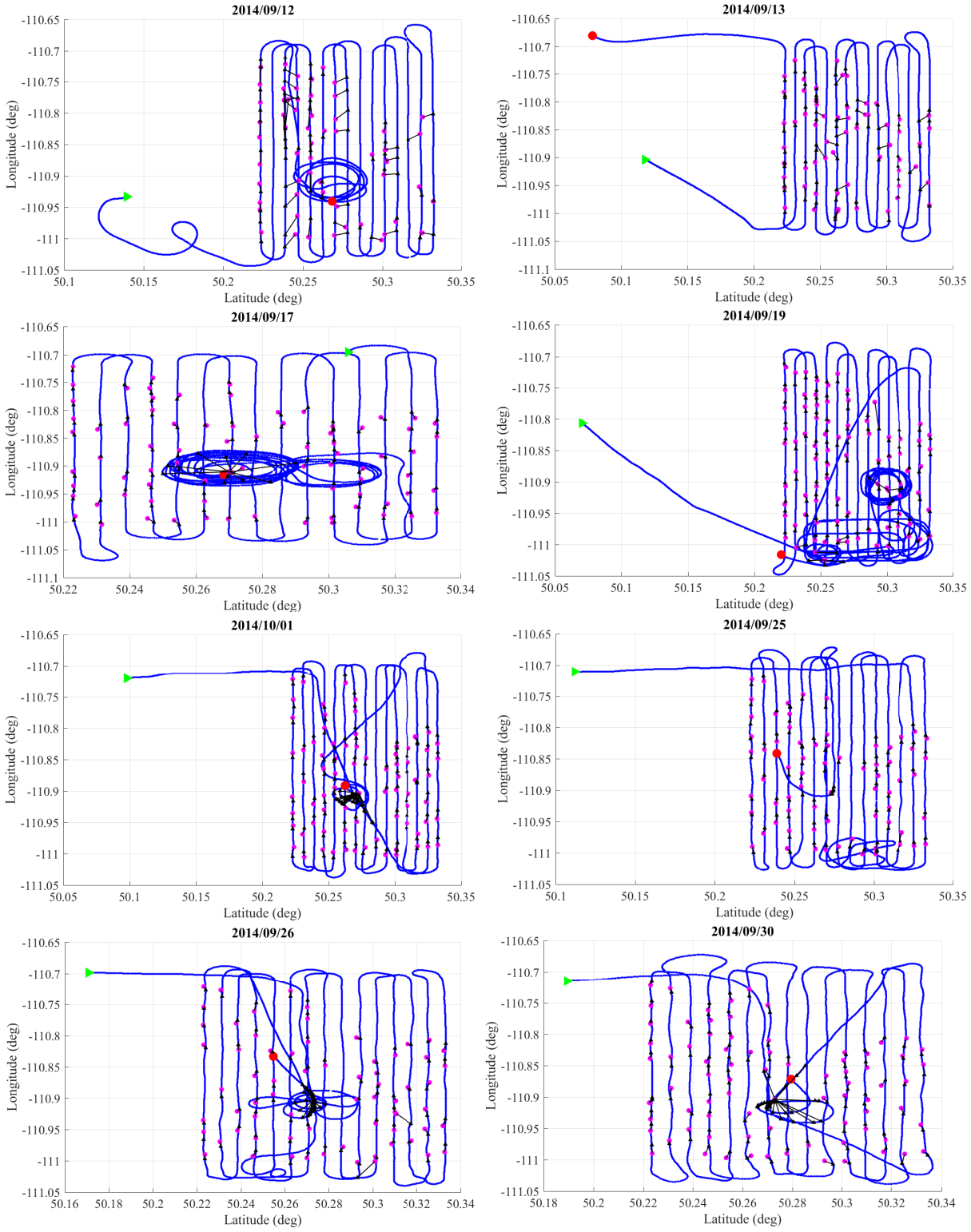

FIGURE 2.6: Aerial search path for each day. Blue contour shows the location of the aircraft over time. The green triangle is the start point, and the red circle is the end point. Black triangles indicate the locations of the airplane at the time of snapshots and magenta circles are the locations of the target in the snapshots. Upon examining the NRC-FRL dataset as to my knowledge, it is unknown why there are some deviations in the aircraft following the track.

2.1.1 Homography

A homography is a transformation that maps one projective plane to another projective plane [33]. A 2D image projection can be related to the scene points for each image by Equation 2.1.

$$x_s = K[R|t]\bar{p}_w = P\bar{p}_w \tag{2.1}$$

Where x_s is the 2D image projection, K is the intrinsic matrix, $[R|t]$ is (rotation+translation) matrix, P is projective matrix and $\bar{p}_w$ is the 3D world point. For convenience, I can align the 3D world coordinate frame with the scene plane, so that $Z = 0$ for all scene points. Under this condition, projection to the image can be modeled by a 3×3 matrix H known as homography (Equation 2.2).

$$x_s = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.2}$$

Since this homography is a 3×3 matrix relating 2D images points in homogeneous coordinates, it has 8 degree of freedom.

$$H = \begin{bmatrix} 1 + h_{00} & h_{01} & h_{02} \\ h_{10} & 1 + h11 & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix} \tag{2.3}$$

Therefore, x' and y' can be defined as:

$$\begin{aligned} x' &= \frac{(1 + h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \\ y' &= \frac{h_{10}x + (1 + h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1} \end{aligned} \tag{2.4}$$

One of the simplest methods for estimating parameter is non-linear least squares. The Jacobian of this transformation is:

$$J = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -y'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix} \tag{2.5}$$

Where $D = h_{20}x + h_{21}y + 1$. The initial guess for the eight unknowns, $h_{00}, \ldots, h_{21}$, can be defined by multiplying both sides of the Equation 2.4 through by the denominator, which yields the linear set of equations:

$$\begin{bmatrix} x' - x \\ y' - y \end{bmatrix} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix} \begin{bmatrix} h_{00} \\ \vdots \\ h_{21} \end{bmatrix} \Leftrightarrow b = AH \tag{2.6}$$

I have four pairs of matched points in this project. Therefore, a unique solution for H is defined. Now I can compute where any point from one projective plane maps onto the second projective plane. There is no need to know the 3D location of the point or camera parameters. For mapping one plane onto the second, I used the skimage.transform module in Python.

2.2 Frame Extraction

I needed to correlate the videos with the log file information to localize targets in the imagery. Challenges include:

- Temporal Shift: Each piece of information in the main CSV files has a time-stamp that indicates when that specific information was recorded. However, the videos are not time-stamped and the beginning of each video does not align exactly with the beginning of each log file.
- Scaling: Given the four corners of the back-projected field of view from the log files, the back-projected field of view can be recreated, but I have noticed that this field of view is much bigger than it should be.
- Video Format: The format of the videos is transport stream (.ts). This is a container format for MPEG that is used frequently by digital broadcasting systems such as digital cable and satellite. It has a very different format from the usual MPEG container. Reading these files in Python is challenging as I could not get consistent size and counts of frames per second. Since I used Python for all our analysis, I had to convert the videos to another format in order to get the exact frame count as well as the exact length of each video. I chose to work with .mp4 as it was easier to manipulate in Python. The transformed videos are not perfectly matched to the log files

In the next experiments I annotated a small subset of our dataset (a few hundreds) in order to define the mapping error considering temporal shift and scaling. Figure 2.7 is a sample of the

mapping and projection of the target location from the ground to image coordinates, without compensating for temporal shift and scaling. The log file ground plane location of the target mapped to the image (red) does not match the actual location of the target in the image (blue).

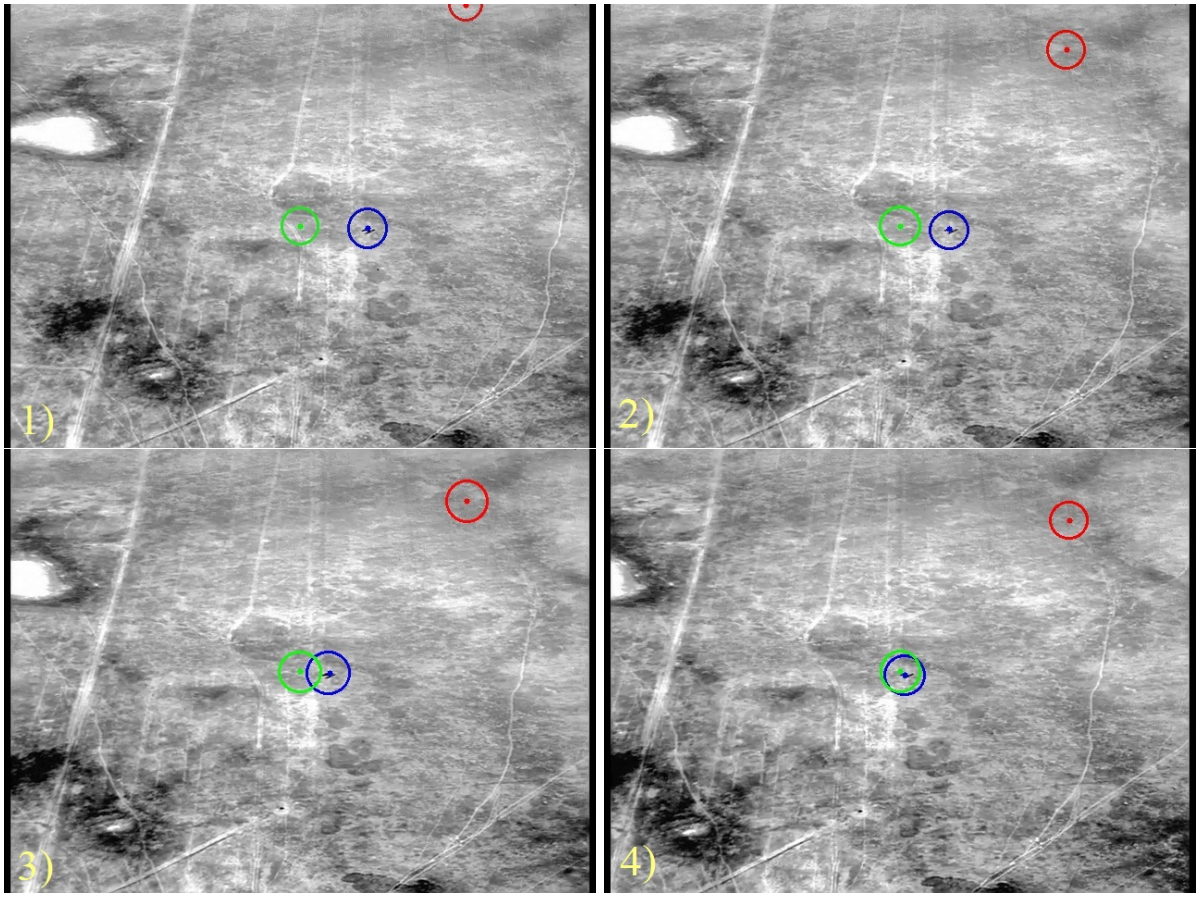

FIGURE 2.7: Sample of mapping from ground to the image plane without compensating for the temporal shift and scaling. These are consecutive frames. The blue circle shows the target on the ground, the green circle shows the center of the image, and the red circle shows the log-file ground-plane location of the target mapped into the image.

Day	Number of Videos	Number of Extracted Frames
2014-09-12	26	41658
2014-09-13	21	35167
2014-09-17	30	51394
2014-09-19	32	47769
2014-09-24	18	30955
2014-09-25	25	44566
2014-09-26	25	44993
2014-09-30	24	42402
2014-10-01	25	43743

TABLE 2.3: Number of videos and total number of frames for each day.

Table 2.3 shows the number of videos and the total number of frames for each day. There are some transitions in the camera's FOV which resulted in blurry and sometimes completely black

or white frames; I have removed these frame from the dataset.

2.3 Scaling the Field of View

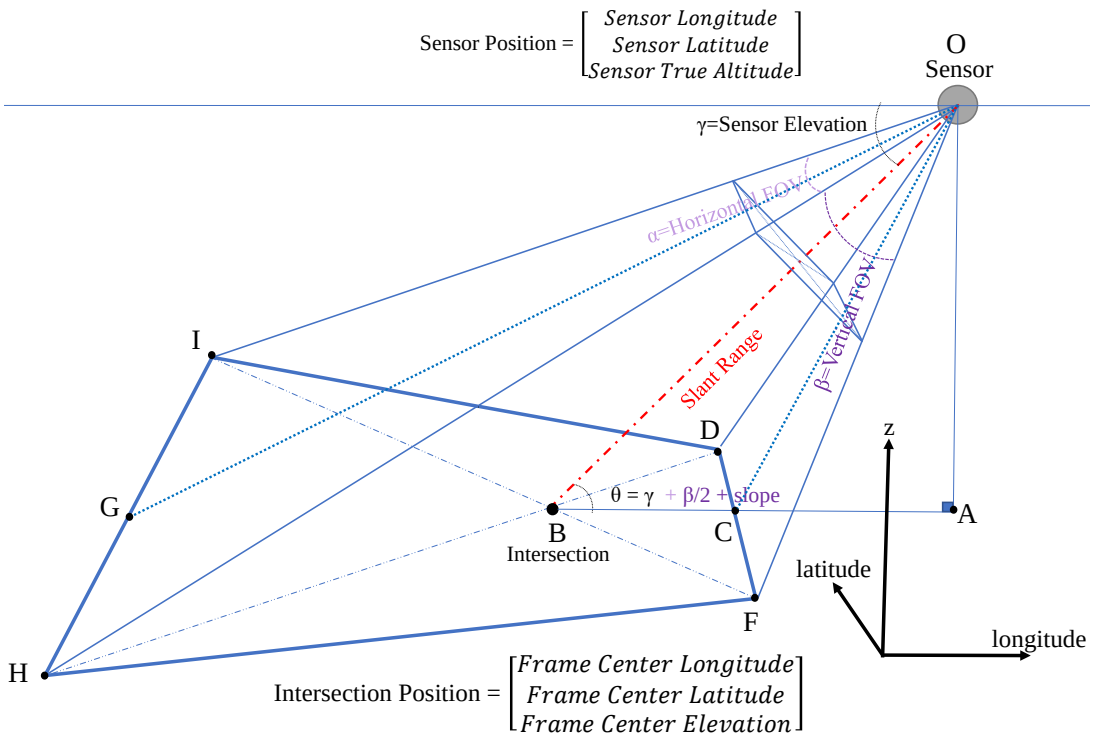

FIGURE 2.8: The polygon represents the sensor's field of view projected on the ground plane.

Given the four corners of the back-projected field of view of the sensor recorded in the log files, the field of view can be recreated. I noticed that the field of view was much bigger than it should be. I therefore decided to use the intersection between the two diagonals of the trapezoid of field of view, the slant range[1], horizontal field of view, vertical field of view, and elevation parameters stored in the log files to create an alternate estimate of the ground-plane field of view.

As shown in Figure 2.8, based on this geometry, I can recompute the new location of the four corners. For example, as in Equation 2.7 and 2.8, I compute the length of BD, in order to find the new corner D. Since the ground in NRC-FRL dataset is fairly flat, I make the assumption that the back-projected field of view happens on a flat plane. This may cause a small error in our application which compared to the noise in the log file information is insignificant.

[1]Slant range is the distance from the middle of the field of view on the ground to the principal point of the sensor

$$
\begin{aligned}
OA &= OB \cdot \cos(\frac{\pi}{2} - \gamma) \\
AC &= OA \cdot \tan(\frac{\pi}{2} - \frac{\beta}{2} - \gamma) \\
AB &= OB \cdot \sin(\frac{\pi}{2} - \gamma) \\
BC &= AB - AC
\end{aligned}
\tag{2.7}
$$

and

$$
\begin{aligned}
OC &= \frac{OA}{\cos(\frac{\pi}{2} - \frac{\beta}{2} - \gamma)} \\
CD &= OC \cdot \tan(\frac{\alpha}{2})
\end{aligned}
\tag{2.8}
$$

I perform the same procedure for all corners. With this calculation, the new polygon will have the same orientation in Earth coordinates at all times. Thus, the last step to complete the creation of the polygon is to rotate it with respect to the yaw or heading angle of the moving airplane as they are recorded with respect to the Earth coordinates. While heading angle is in the log file, I decided instead to use the location of the intersection point to define the heading angle and avoid using the other noisy data in the log files. Given the (latitude and longitude) position of the aircraft at two successive times, I can compute the heading angle of the airplane using the Python package Geodesic.Wgs84.Inverse.

2.4 Temporal Shift

When a target was noticed by the human operator, they zoomed in on the target for a few seconds. Information in the log file identifies the camera's horizontal and vertical fields of view, which makes it possible to figure out roughly when the operator noticed the target and zoomed in. Due to various factors, there is an unknown temporal shift between this log file event and the corresponding frame in the video. To reduce this temporal shift and to create a better match between the log files and the video frames, I have tried two methods, one manual and one automatic.

In the first method, I used the camera's horizontal and vertical fields of view in the log file and looked for the time-stamp in which the zoom occurred. Meanwhile, I manually identified frames in the video where the human operator zoomed in. There is a frame of the video for each time-stamp in the log file. Therefore, the difference between the number of the frame of

Video/Day	09-12	09-13	09-17	09-19	09-24	09-25	09-26	09-30	10-01
0	0	18	0	28	6	0	8	13	0
1	26	25	14	28	13	10	11	14	0
2	31	25	15	21	N/T	19	6	10	41
3	24	20	4	20	22	15	11	19	14
4	27	32	9	19	N/T	6	19	7	8
5	22	28	16	N/T	12	25	13	18	5
6	22	19	4	18	N/T	13	19	20	15
7	29	21	6	32	N/T	20	15	12	21
8	32	13	15	17	N/T	10	14	10	18
9	33	31	3	20	N/T	15	11	8	20
10	29	29	6	31	N/T	15	7	20	12
11	24	28	18	19	N/T	23	6	10	18
12	20	28	8	19	4	15	7	16	9
13	20	N/T	18	18	21	17	6	16	13
14	34	28	7	28	17	8	4	21	16
15	31	30	14	28	20	9	5	10	10
16	28	19	11	20	15	16	15	21	9
17	31	27	3	30	N/T	18	9	12	13
18	28	22	12	18		6	8	17	6
19	20	25	5	20		20	16	12	19
20	33	0	0	23		0	9	5	11
21	25		13	19		0	5	14	5
22	N/T		18	22		6	19	11	14
23	N/T		10	22		0	16	N/T	0
24	N/T		6	31		0	16	N/T	0
25	N/T		N/T	25				N/T	
26			6	25					
27			17	17					
28			N/T	19					
29			N/T	23					
30				N/T					
31				N/T					

TABLE 2.4: Manually-estimated frames shifts. N/T means no target was seen in this video so I could not estimate the shift and, as it does not contain any target, it is not considered in our analysis.

video corresponds to the time-stamp when the operator began to zoom in on the target and the number of the frames in the video where zoom occurred, defines the temporal shift. Considering this shift in frames, I can match the log file information to the extracted frames more accurately. Table 2.4 shows manually extracted shifts for different days and videos. As shown, the temporal shift is different for each day and video.

The location of each target placed on the ground (in latitude and longitude) was recorded by NRC-FRL staff. Knowing the latitude and longitude of each frame's field of view, I can find frames with at least one target located within its field of view. I used homography 2.1.1 to

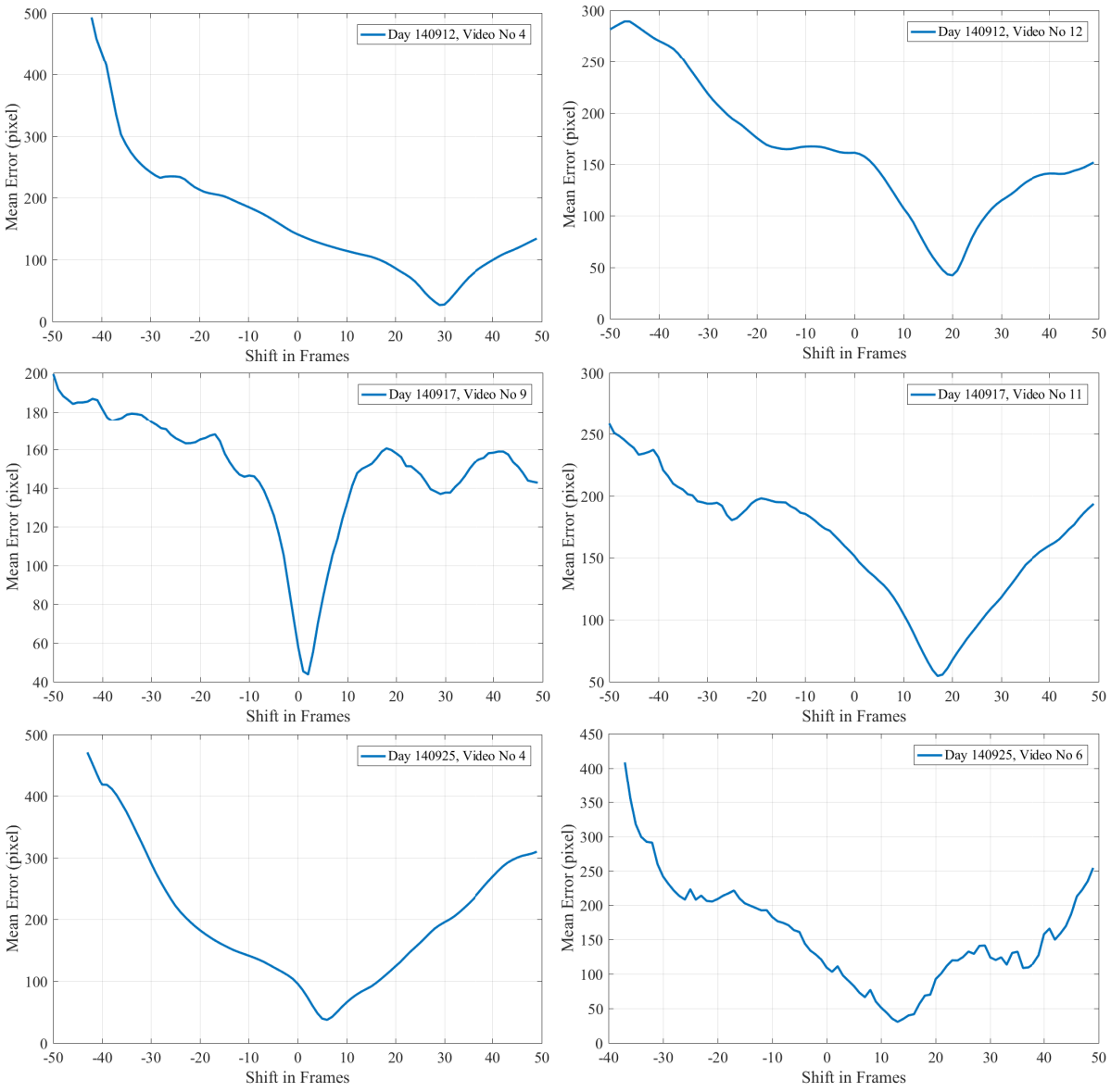

FIGURE 2.9: Mean error between the target in the image and the projected location of the target into the image, based on the ground location and the ground-to-sensor homography for six different example cases. The frame shift with minimum error matches the manual estimates in Table 2.4.

identify the target's location in pixels on the corresponding image. Since I have also manually localized each target appearing in an image frame for a small subset of our dataset, I can find the error in pixels between the target location estimate derived by projecting the ground plane target location and the manually annotated image location of the target. Using inverse homography, I can map the location of the labeled target on the image to the ground and find the error in meters between the target location recorded by the NRC-FRL staff and the manually annotated location. In the second method, to confirm the accuracy of the result, I automatically identified the mean error between the manually annotated location of the target and target location estimate for different shifts in the range of -50 to $+50$ frames. I then

estimate the shift as the shift that minimizes the error. As depicted in Figure 2.9 and Table 2.4, the results from these two methods match.

2.5 Labeling

Day	Number of labeled frames
2014-09-12	3113
2014-09-13	3154
2014-09-17	5349
2014-09-19	5310
2014-09-24	168
2014-09-25	3897
2014-09-26	4346
2014-09-30	5883
2014-10-01	6419

TABLE 2.5: Labeled frames for each day.

For labeling the whole dataset, I considered 400,000 images to find images with a target. I considered the scaling issue and the temporal shift in finding images with a target. In the original SAR experiment, when a target was observed, the human operator took a picture of it and recorded the current position of the sensor in longitude and latitude along with a time-stamp. Using this time-stamp makes the labeling process much easier since I can estimate where to look for possible frames with targets among all the frames of the videos. As shown in Table 2.5, using MATLAB, I was able to label and annotate about 40,000 images containing a target.

2.6 Dataset Partitioning

To train the classifier, I divided the datatset into 3 main parts:

- East side for training
- Northwest quadrant for validation
- Southwest quadrant for test

I chose the longitude and latitude boundaries to produce an almost equal number of targets on the ground in the training set versus the validation and test sets. There are many zoomed-in annotated images in the dataset. Since I am using it for the purpose of search and rescue, I needed to eliminate the zoomed-in frames. Considering this, I have in total 20,401 negative images and 20,403 positive images in the dataset. The number of positive frames identified for

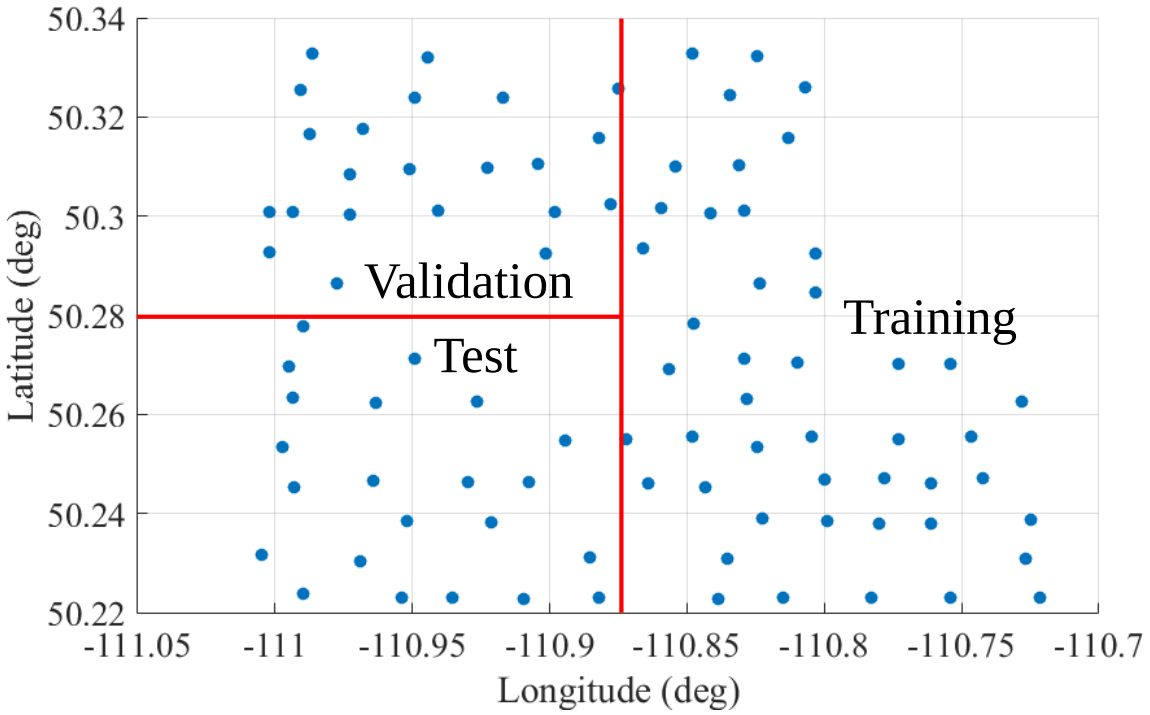

FIGURE 2.10: Data partitioning.

each day is matched by an equal number of randomly selected negative frames from the same day. As in Table 2.6, there are 9,904 positive training images and 10,626 negative training images. Also there are 3,994 positive validation images and 3,754 negative validation images.

	Training	Validation	Test
Positive	9,904	3,994	6,505
Negative	10,626	3,754	6,021

TABLE 2.6: Final dataset.

Figure 2.11 shows the distribution of target length and aspect ratios for about 400 targets randomly sampled from our dataset. To compute this distribution, I manually annotated the noise, tail and wingtips of the aircraft. Figure 2.12 indicates that the aspect ratio is close to one, so that square detection windows could potentially be used.

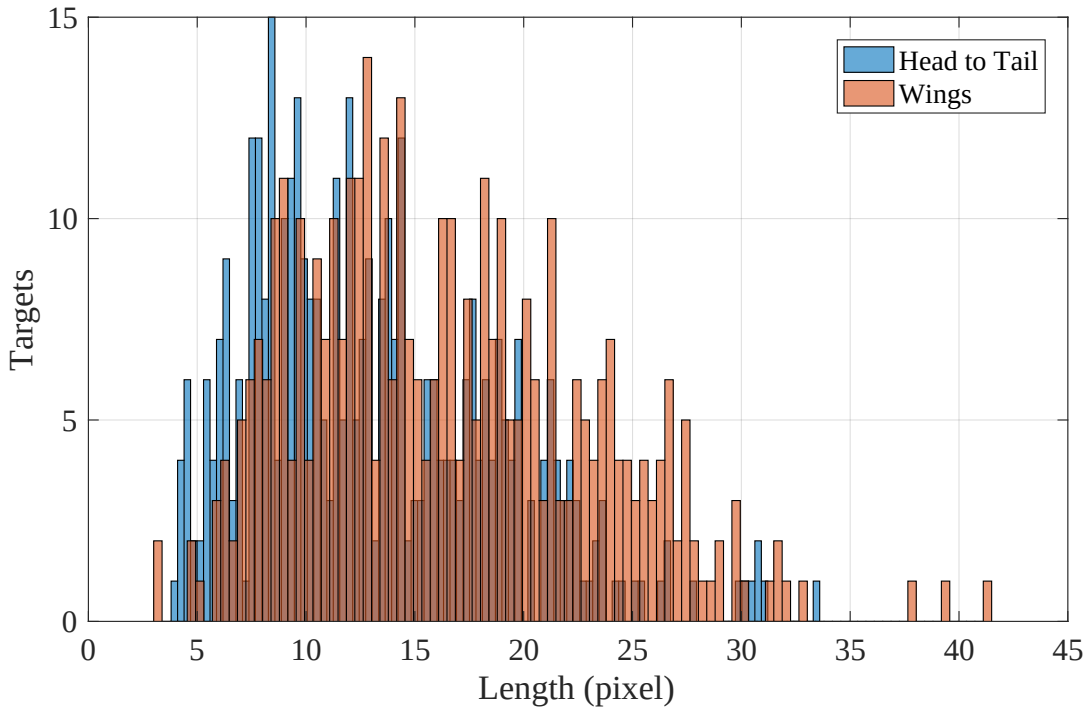

FIGURE 2.11: Dimension's histograms for 400 samples. Most of the targets have smaller dimensions than of 35 pixels.

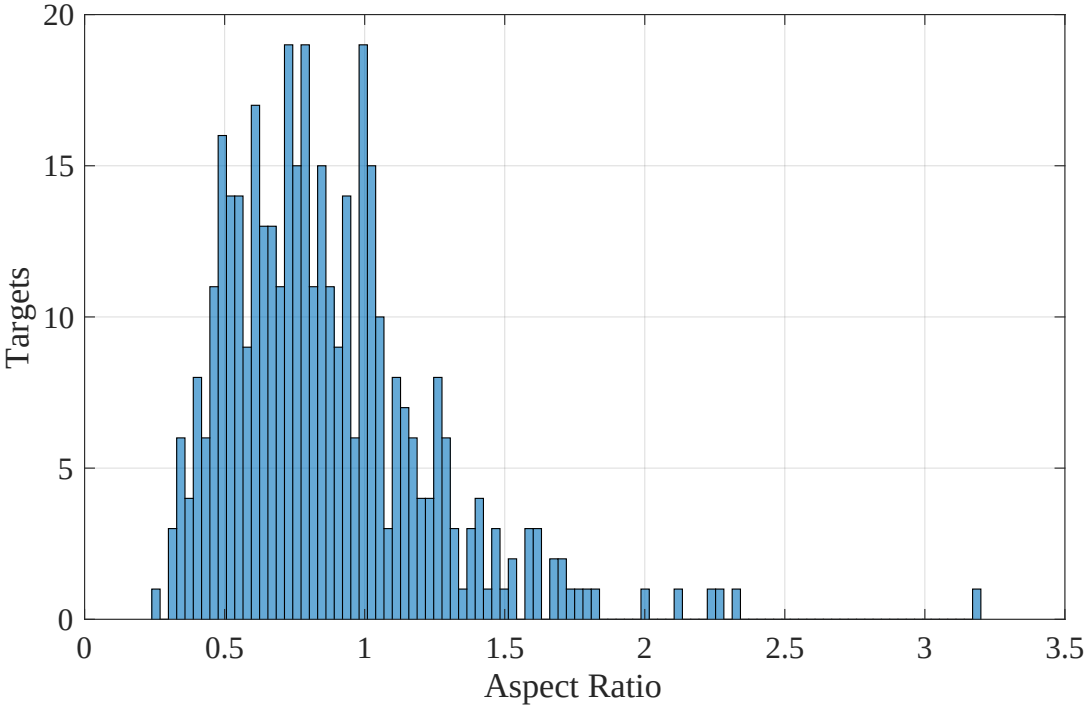

FIGURE 2.12: Aspect ratios histograms of head-tail to wings' length.

Chapter 3

Assisted Target Detection

In this chapter I analyze the results of the three different methods introduced in the previous chapter – Matched Filters (MF), Histogram of Oriented Gradients (HOG), and Deep Learning – and compare their performance as detectors. We will see that the target detection performance of the first two methods, MF and HOG, is significantly worse than that of the deep learning method. Implementing these algorithms, especially deep learning, is computationally intensive. I used just the CPU for MF and HOG, and employed a GPU for deep learning. I used Ubuntu 16.04.6 running on an Intel ® Core ™ i7-4770(3.4 GHz) CPU with an NVIDIA GeForce GTX 1080 Ti GPU.

One of the common ways to visualize the performance of detection algorithms is a precision-recall curve (PR curve). I can plot this PR curve by having the confidence score and the true label for each detection. By varying the threshold on confidence continuously, I obtain a curve of *precision* and *recall* values that I can graph to compare and analyze the performance of each detector.

After the implementation of the non-maximum suppression (NMS), I can create an array of confidence scores and true labels and sort that array by the confidence scores, which enables me to define the true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). For instance, for the case shown in Figure 3.1, after defining a threshold, all detections with the label *1* (positive images) above the threshold are considered as TP, and all detections with the label *0* (negative images) above the threshold are considered as FP. Detections with the label *0* below the threshold are TN, and those with the label *1* under the threshold are FN. Having these values, I can calculate the *precision* and the *recall*, where $precision = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$ and $recall = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$.

In this project I need *recall* to be relatively high, as even a single FN might mean that I miss finding the airplane. However, if there is only a single target, then even a low *precision* might

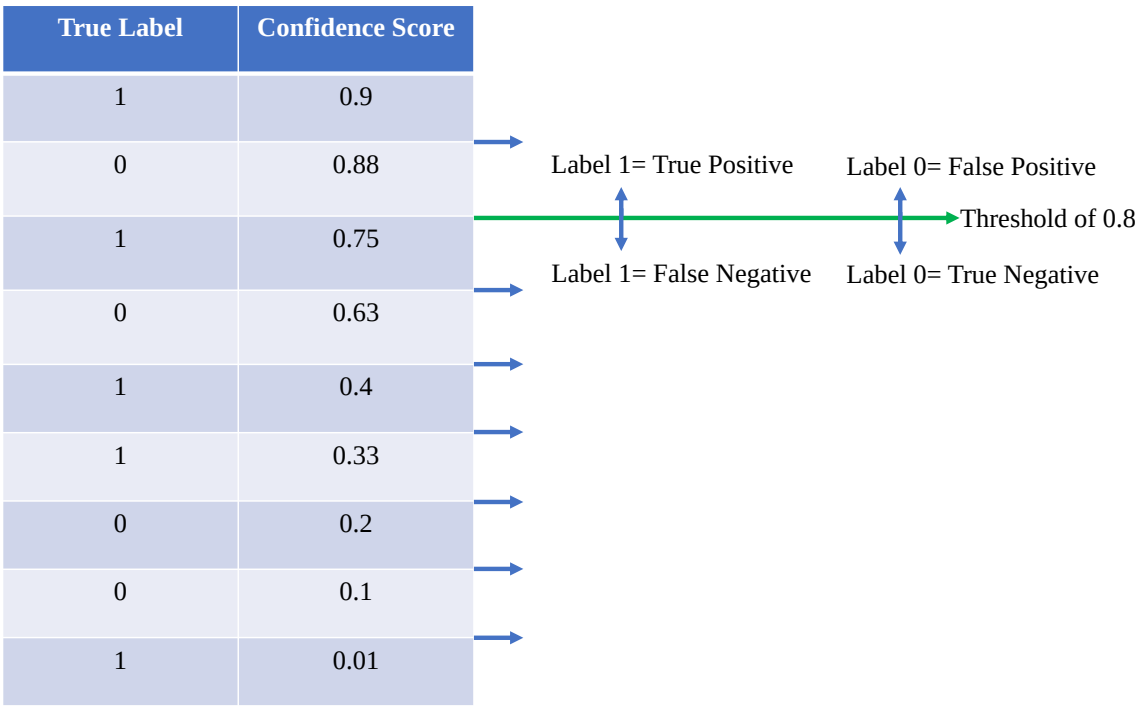

FIGURE 3.1: An example of the confidence scores array and their true labels. By defining a threshold I can define the true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN).

be tolerable. For example, if there is just one airplane, then a search operator might be able to filter through hundreds of false positives. In that case, the *precision* would be quite low. Since *precision* is largely defined by the ratio TP/FP, it is very sensitive to the balance of the dataset. For example, if the dataset has 1 positive image and 1,000 negative images, even a moderate FP will result in low *precision*.

Therefore, below I describe two ways of testing the MF, HOG, and deep neural networks (DNN) methods. First, I test the algorithm on equal numbers of positive and negative images to see how it performs with a more approximately balanced dataset (part (A) in section 3.1.1),. Second, I test the algorithms as they would be used in the real world (part (B) in section 3.1.1), with more negative images than positive images.

3.1 Matched Filters

3.1.1 Approach

The basic idea in MF is to create a filter or a template based on the positive and negative images of the dataset as described in section 1.3. By computing the inner product of this template across either (A) image patches from the dataset that are the same size as the template or, (B) each

of the images of size 640×480 pixels in the dataset, I can determine if either (A) any of the image patches or (B) any parts of that image have a similar structure to the template.

For the sliding window method, I used a square window that moves across the image in 2 pixel steps in horizontal and vertical directions. Positive cases are defined as cases where the center of the sliding window has a distance less than half of the average size of an actual target in the dataset.

Taking the inner product of the template across the whole image as in (B) will produce many more negative images than positive ones. For instance, for a sliding window of size 20×20 with a stride of 2 pixels and an image size of 480×640, a positive image will be divided into 76,800 image patches, of which fewer than 30 are true positives. This means that for a validation set of, for example, 100 images (half positive images and half negative images), I have fewer than 3,000 positive image patches and 7,677,000 negative image patches. This imbalance in the positive and negative image patches reduces the *precision*.

I used the training partition of the dataset to construct the template for the MF - approximately 20,000 images (about 10,000 positive images and 10,000 negative images). I evaluated four template sizes: 10×10, 20×20, 40×40, and 80×80 pixels. I also implemented a method to minimize variation in target size by projecting to ground plane coordinates. This is described under the section 3.1.2 "Geometry". The template is defined as the difference of the average negative image from the average positive image. Figure 3.2 shows the average target and average background, and Figure 3.3 shows the template.

In the case of (A) above, I extract positive image patches from the positive images and negative image patches using the same procedure described above in constructing the template. This provides about 4,000 positive image patches and 4,000 negative image patches that are the same size as the template. I then compute the inner product for each of these 8,000 image patches. In the case of (B) above, I take the inner product of the template with each image from the validation dataset (50 positive, 50 negative images), moving the template in a sliding window with stride 5 across the image.

3.1.2 Geometry

Along with each frame of video in the dataset, I have information about the searching airplane, including horizontal and vertical field of view, sensor elevation, and slant range[1]. I use this information to equalize the expected target size in the sliding window approach. As a sliding window moving across an image, each image patch is re-sized based on the distance of the patch center from the ground to the camera. For example, in Figure 3.4, the center of the sliding

[1] Slant range is the distance from the ground-plane projection of the principle point to the center of the optical sensor

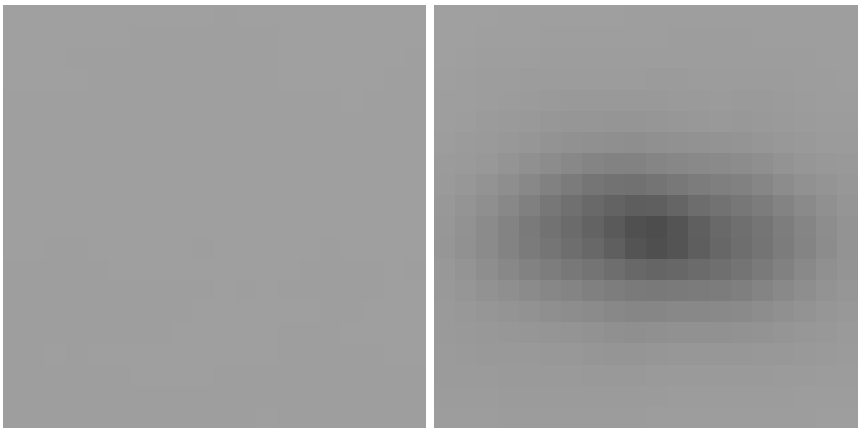

FIGURE 3.2: On the left is the average of the images without a target, and on the right is the average of the images with targets for the training dataset. Window size = 40×40 pixels.

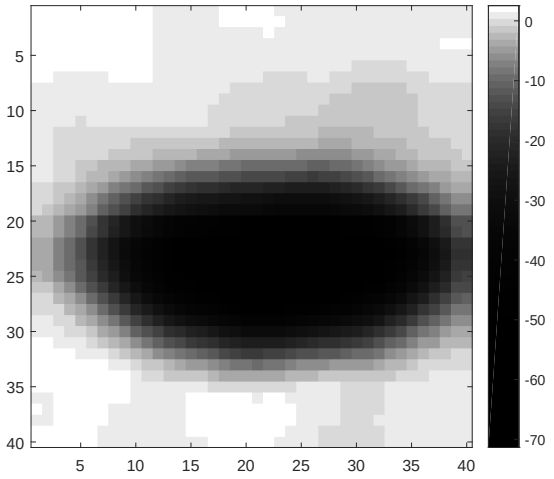

FIGURE 3.3: MF template.

window corresponds to a point in the real world E with distance to the center of the camera denoted as AE. D is the closest point on trapezoid's height in the middle, to the point E.

Since I know the sensor elevation angle (γ), I can use the right-angled triangle property of $\triangle ABC$ and calculate AB and AD:

$$AB = \text{slant range} * sin(\gamma) \tag{3.1}$$

$$AD = \frac{AB}{\sin(\phi)} \tag{3.2}$$

Based on whether the location of the sliding window is above or below the center of the image, I can define the angle ϕ as:

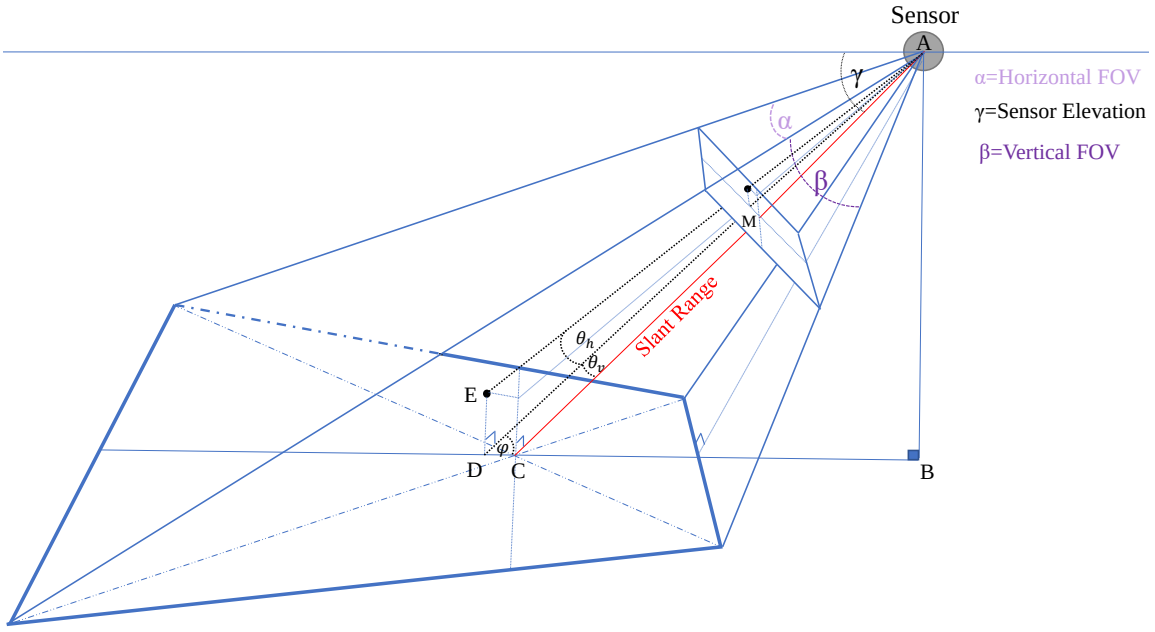

FIGURE 3.4: Each point in the image corresponds to a point on the ground in the field of view. I use the corrected field of view as described in the chapter "Dataset". Sections 2.3 and 2.4.

$$\phi = \begin{cases} \gamma - \theta_v, & \text{Below} \\ \gamma + \theta_v, & \text{Above} \end{cases}$$

where θ_v is the angle from the camera to the center of the sliding window in the y direction as shown in Figure 3.5. The angle from the camera to the sliding window center in the x direction is denoted as θ_h, so I can define AE as:

$$AE = \frac{AD}{\cos(\theta_h)} \tag{3.3}$$

The focal length of the camera installed on the searching airplane (f) is $40mm$ and each pixel in the camera is $20\mu m$. θ_v and θ_h based on the focal length are defined in Equations 3.4 and 3.5.

$$\theta_v = tan^{-1}\frac{y}{f} \tag{3.4}$$

$$\theta_h = tan^{-1}\frac{x}{f} \tag{3.5}$$

Since the dimension of the target is proportional to the inverse of its distance to the center of the camera, the right window size for each image patch is:

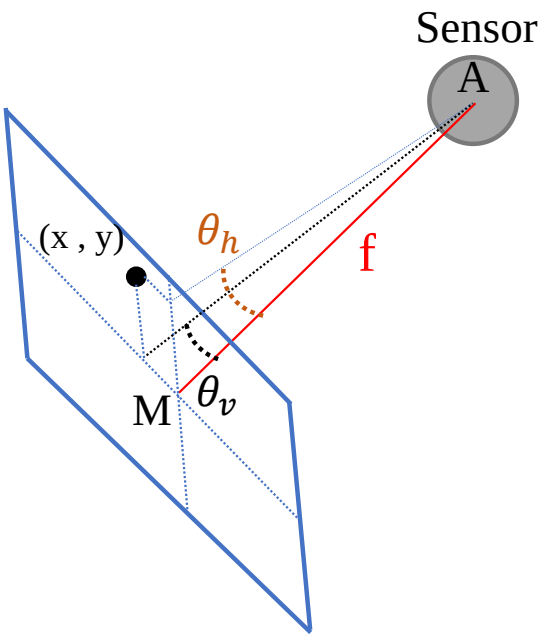

FIGURE 3.5: θ_h and θ_v are angles from the center of the camera to the center of the sliding window in x and y directions.

$$\text{window size} = \frac{C}{AE} \tag{3.6}$$

where C is a constant over all pixels and slant ranges. Figure 3.6 shows the distribution of slant ranges in our dataset. It ranges mainly from $500m$ to about $2200m$. I ran an experiment to identify the constant C that provides the best performance with MF and HOG methods. These results are described in the "Results" section for each method. Figure 3.7 shows some samples of the original cropped image patches with a window size of 40×40 pixels versus image patches with window size modified according to the Equation 3.6. The method clearly helps to equalize the target size.

3.1.3 Results

Figure 3.8 shows the probability distributions of the inner product calculated by matching the template shown in Figure 3.3 with training image patches and Figure 3.9 shows the probability distribution for the patch-based validation set extracted by method (A)(section 3.1.1). Figure 3.10 shows the same distributions for patches extracted with method (B) after implementing the non-maximum suppression. The positive distribution in method (A) is based on approximately 5,000 different targets. There is no clear margin between the two sets of scores of the positive and negative images; consequently, a threshold that would provide for good detection performance

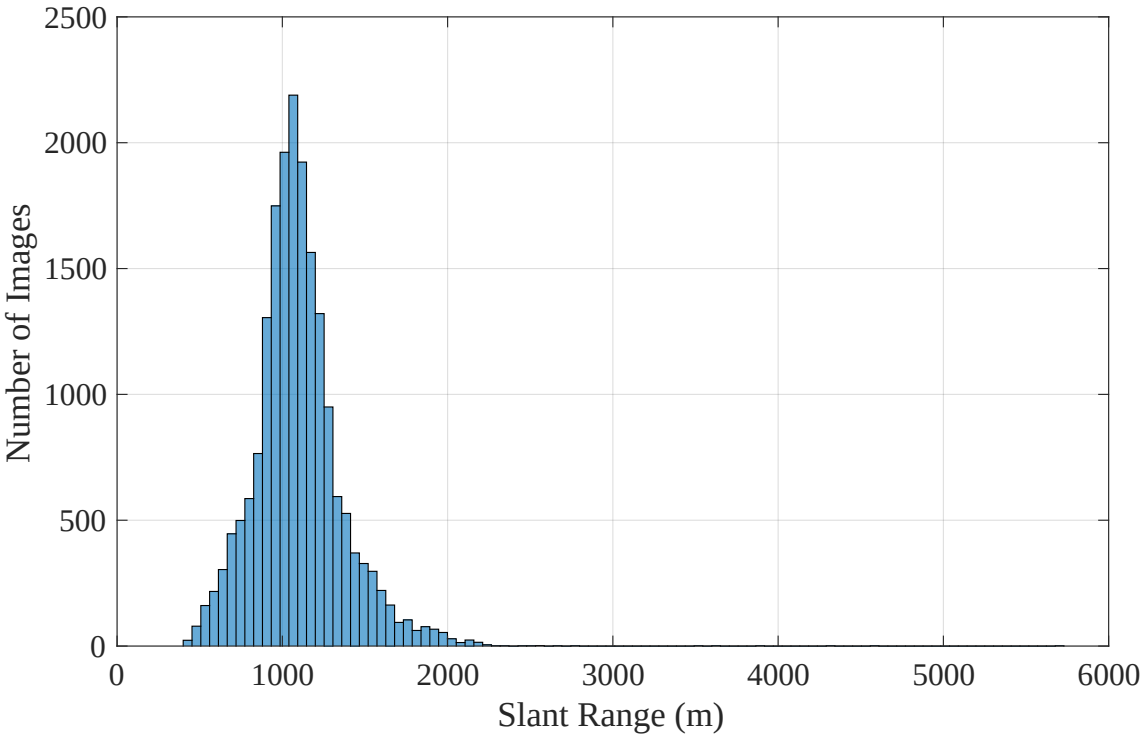

FIGURE 3.6: Slant range histogram shows the distance from the center of the camera to the center of the image on the ground for each image in our dataset. For our dataset, this distance ranges from roughly $500m$ to the roughly $2000m$.

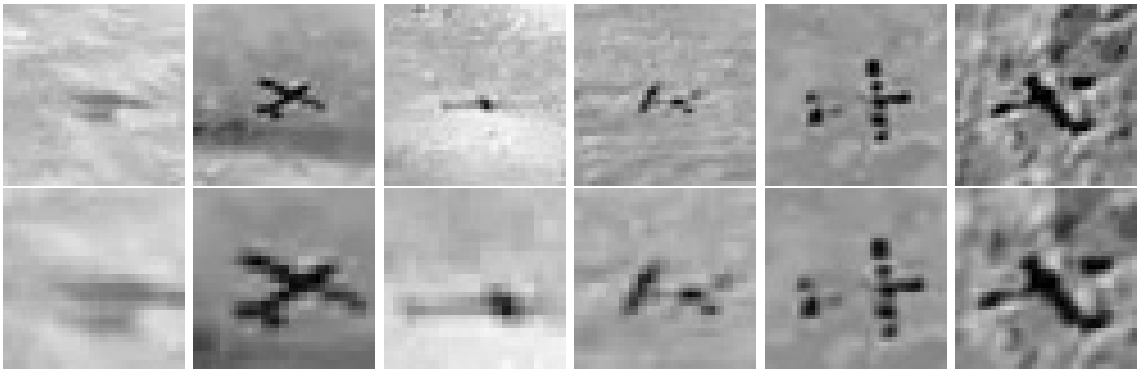

FIGURE 3.7: Upper row are samples of targets shown in original size of the sliding window, 40×40 pixels. Lower row are new cropped patches based on their distance to the camera.

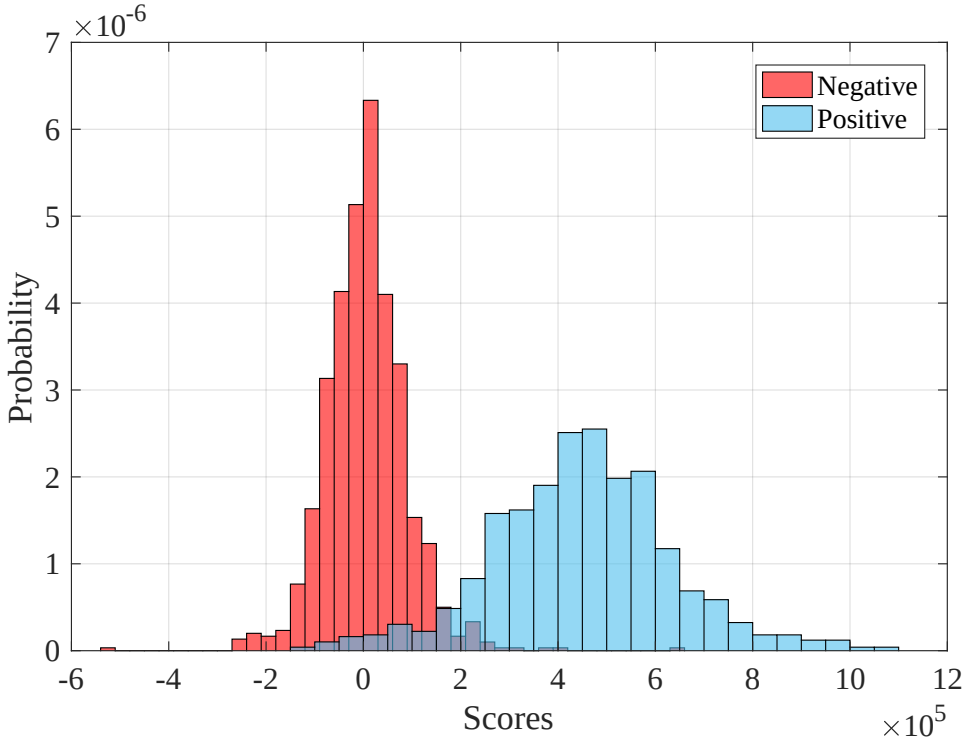

FIGURE 3.8: Probability distribution of the inner product of the template with positive and negative image patches of size 40×40 pixels based on 10,000 image patches from the training set extracted using Method (A)(Section 3.1.1).

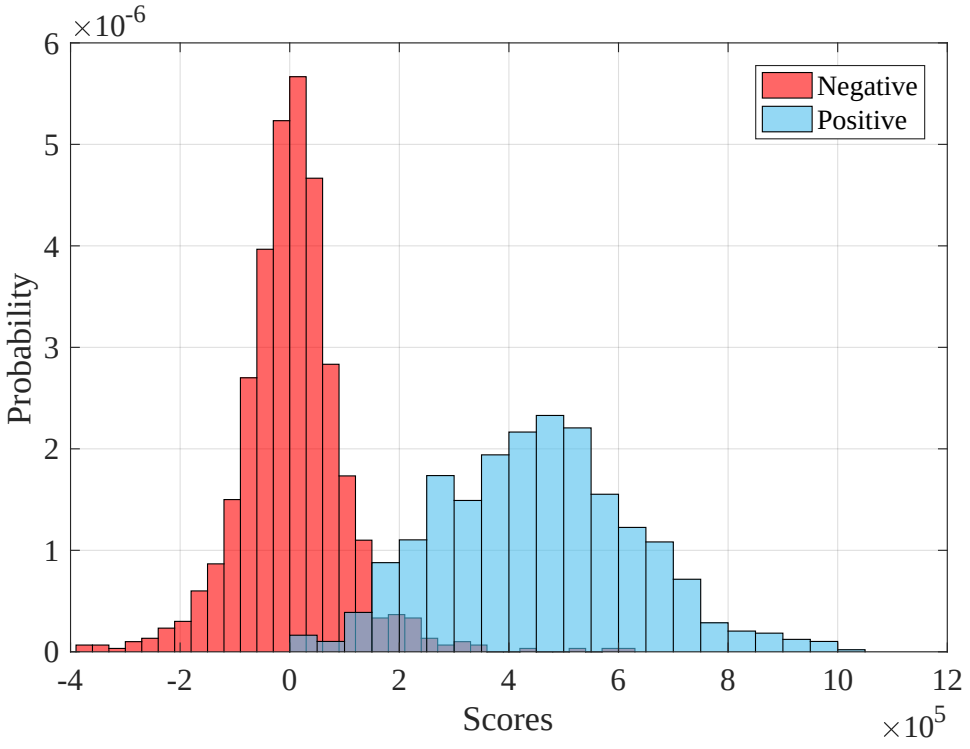

FIGURE 3.9: Probability distribution of the inner product of the template with positive and negative image patches of size 40×40 pixels based on 10,000 image patches from the patch-based validation set extracted using Method (A)(Section 3.1.1).

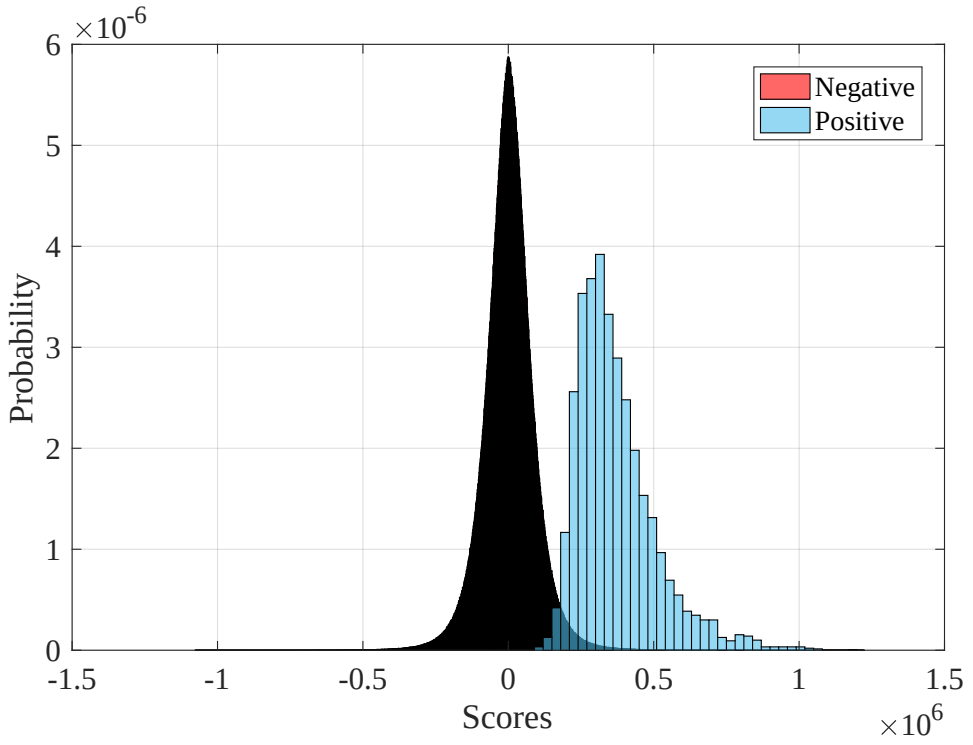

FIGURE 3.10: Probability distribution of the inner product of the template with positive and negative image patches of size 40×40 pixels based on images of size 480×640 pixels from the validation set using Method (B)(Section 3.1.1).

cannot be chosen. In fact, we can see from these two distributions that we can obtain high *recall* or high *precision*, but not both.

Figure 3.11 shows the Receiver Operating Characteristic curves (ROC curves) of MF algorithm performance for different sliding windows on image patches, method (A). To assess the detector performance, I plotted PR curves as well as ROC curves. ROC curves use a different measure that allows for a different comparison of the algorithm's performance across the datasets.

In ROC curves, the true positive rate (which is as the same as *recall*) is plotted against the false positive rate, where the true positive rate is defined as $\frac{\text{TP}}{\text{TP+FN}}$ and the false positive rate is defined as $\frac{\text{FP}}{\text{FP+TN}}$. The true positive rate is a measure of how many of the positive samples have been correctly identified, while the false positive rate is a measure of how many of the negative samples have been incorrectly identified.

In Figure 3.12, the MF algorithm PR curve shows that MF algorithm performs very poorly on the image validation set. The precision over the set of validation images is very low. Among all methods, the Geometry method has slightly better performance. I chose the constant $C = 28,000$ as it has a better performance on the validation set. Figure 3.13 shows MF performance over different overlaps in non-maximum suppression for the Geometry method. I chose the overlap 0 as it has a better performance not only for the Geometry method but also for MF

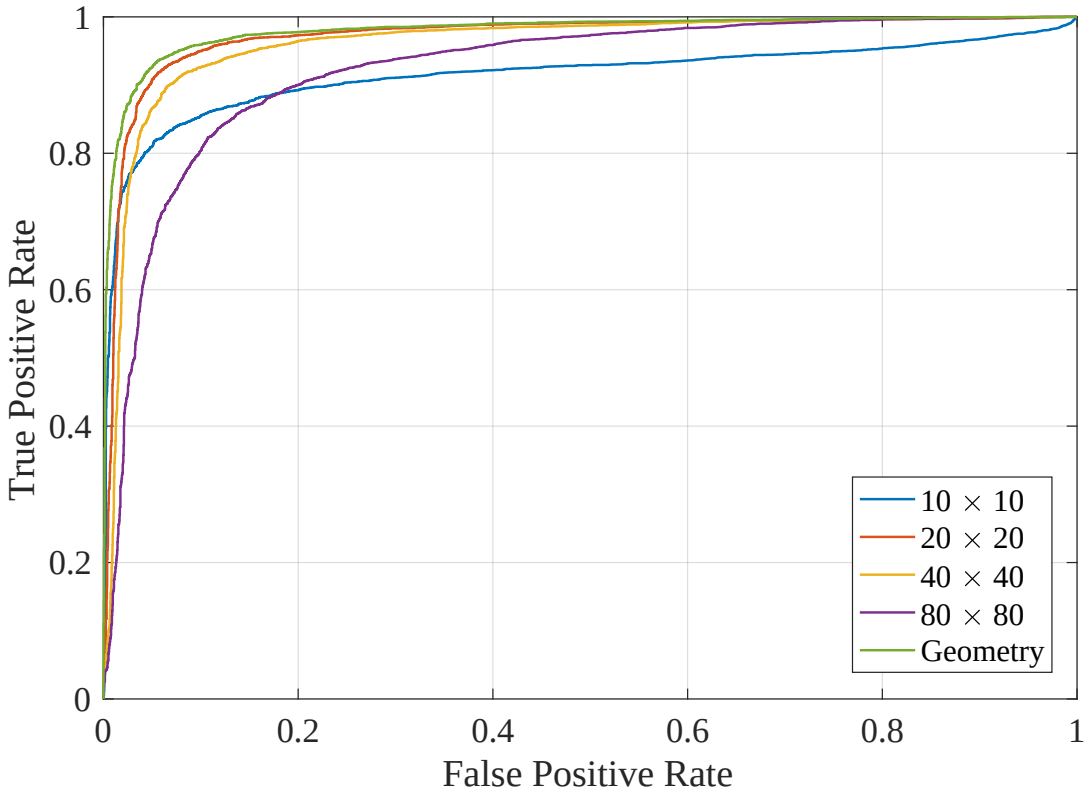

FIGURE 3.11: ROC curve of MF with different sliding window sizes on image patches.

with different window sizes. The MF algorithm as a detector is very dependent on the filter or template that I use in the algorithm. We can clearly see the airplanes in false negatives (Figure 3.14), but they differ from the template, mostly in orientation and luminance. Moreover, miss hits can occur due to spatial shifts, missing parts, and low contrast. We can also see that almost all of the most confident false positives have a vague resemblance to the template, as they are mostly dark in the middle and lighter in the background. Figure 3.18 shows a few results of the MF method on the validation set, after non-maximum suppression. For the Geometry method, it took 38.25 seconds to create the template and 11.58 seconds for implementing the algorithm on the each validation image.

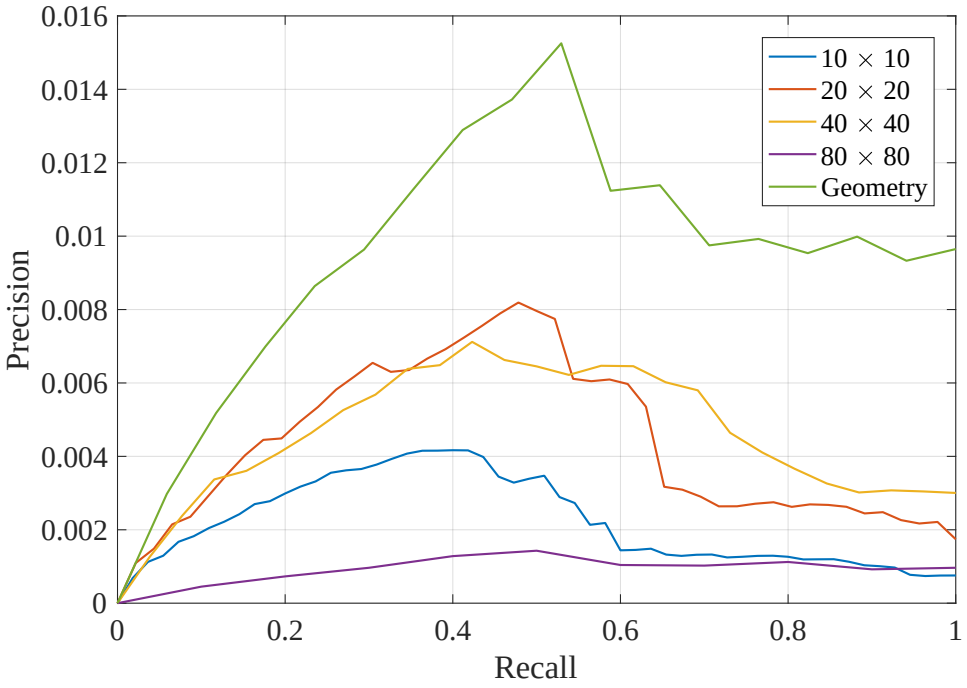

FIGURE 3.12: Precision-Recall curve of MF over the IR images with different sliding window sizes.

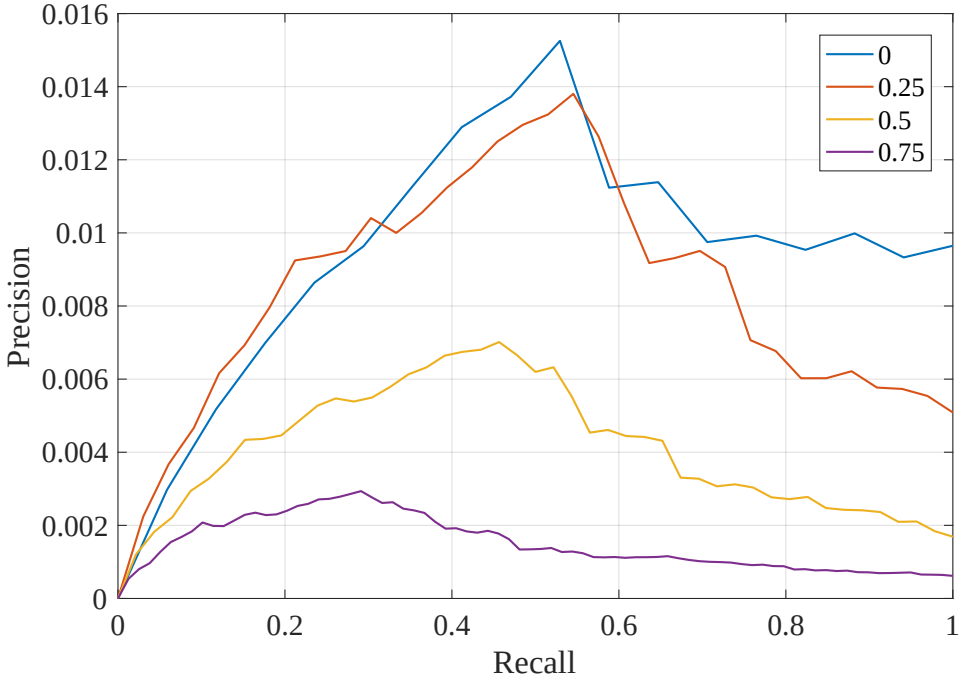

FIGURE 3.13: Precision-Recall curve of MF with Geometry method over the IR images with different overlaps in non-maximum suppression.

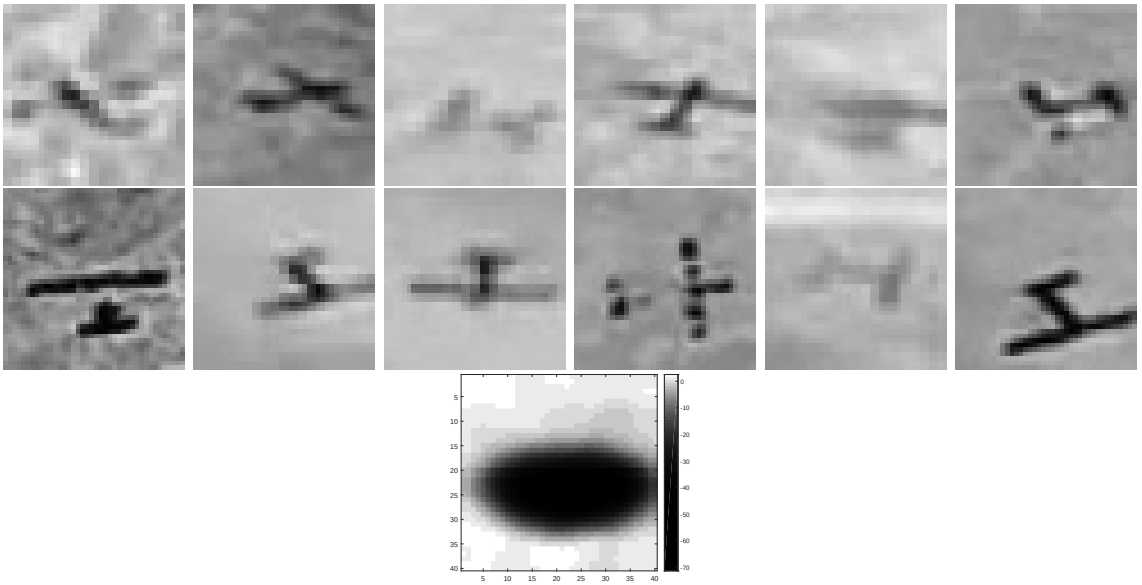

FIGURE 3.14: False negative patches for MF algorithm. The template that was used is in the bottom row.

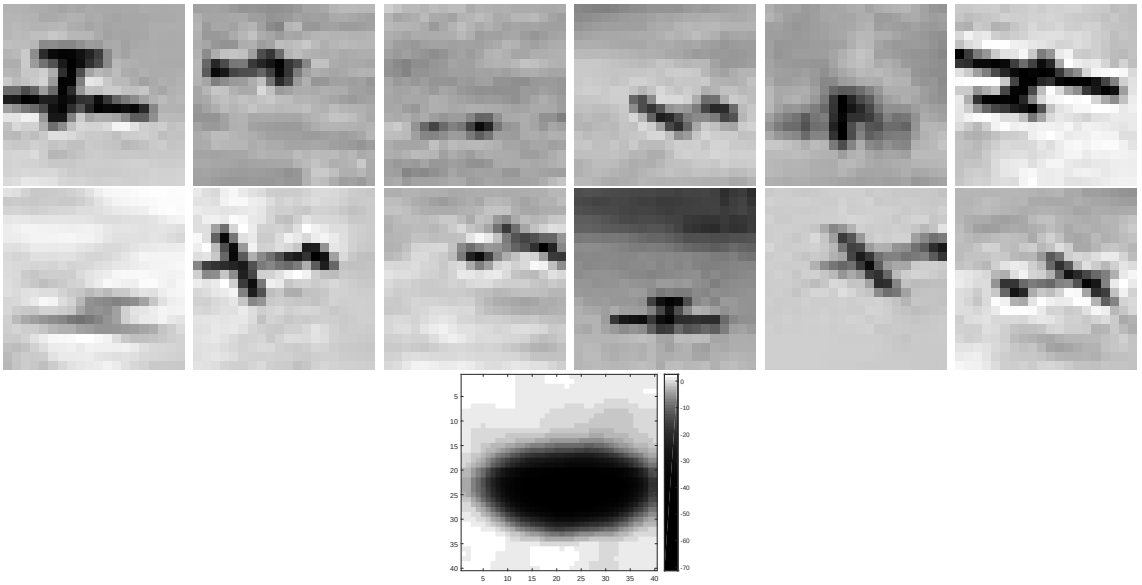

FIGURE 3.15: True positive patches for MF algorithm. The template that was used is in the bottom row.

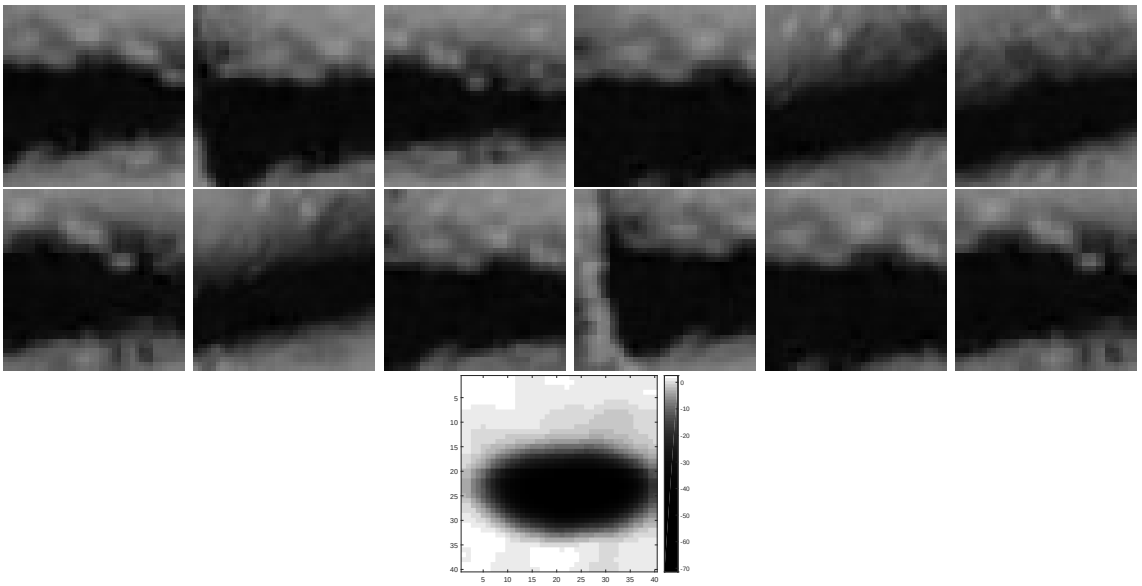

FIGURE 3.16: False positive patches for MF algorithm. The template that was used is in the bottom row.

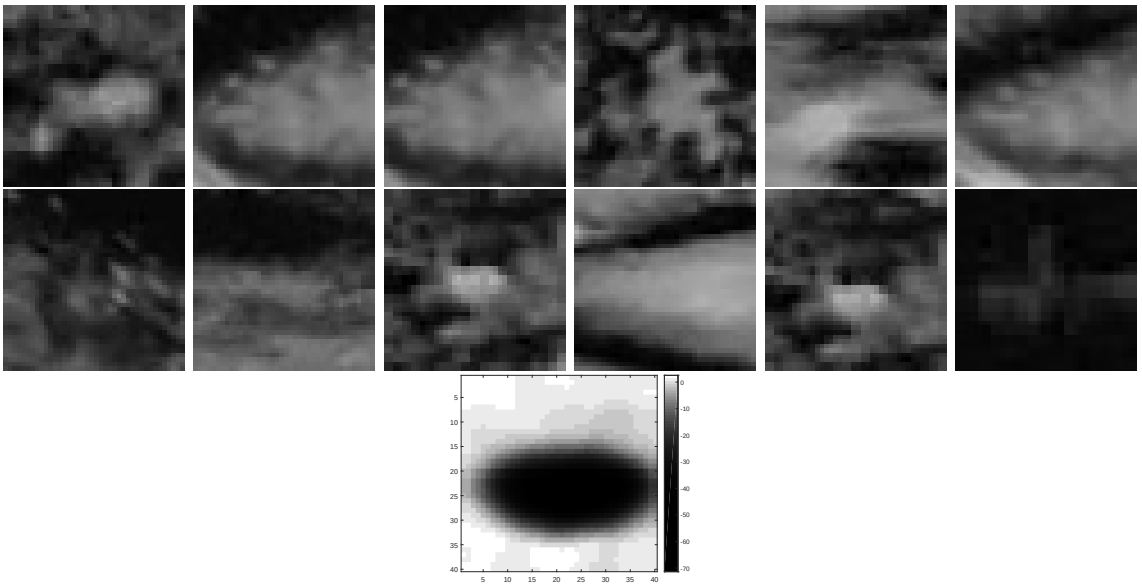

FIGURE 3.17: True negative patches for MF algorithm. The template that was used is in the bottom row.

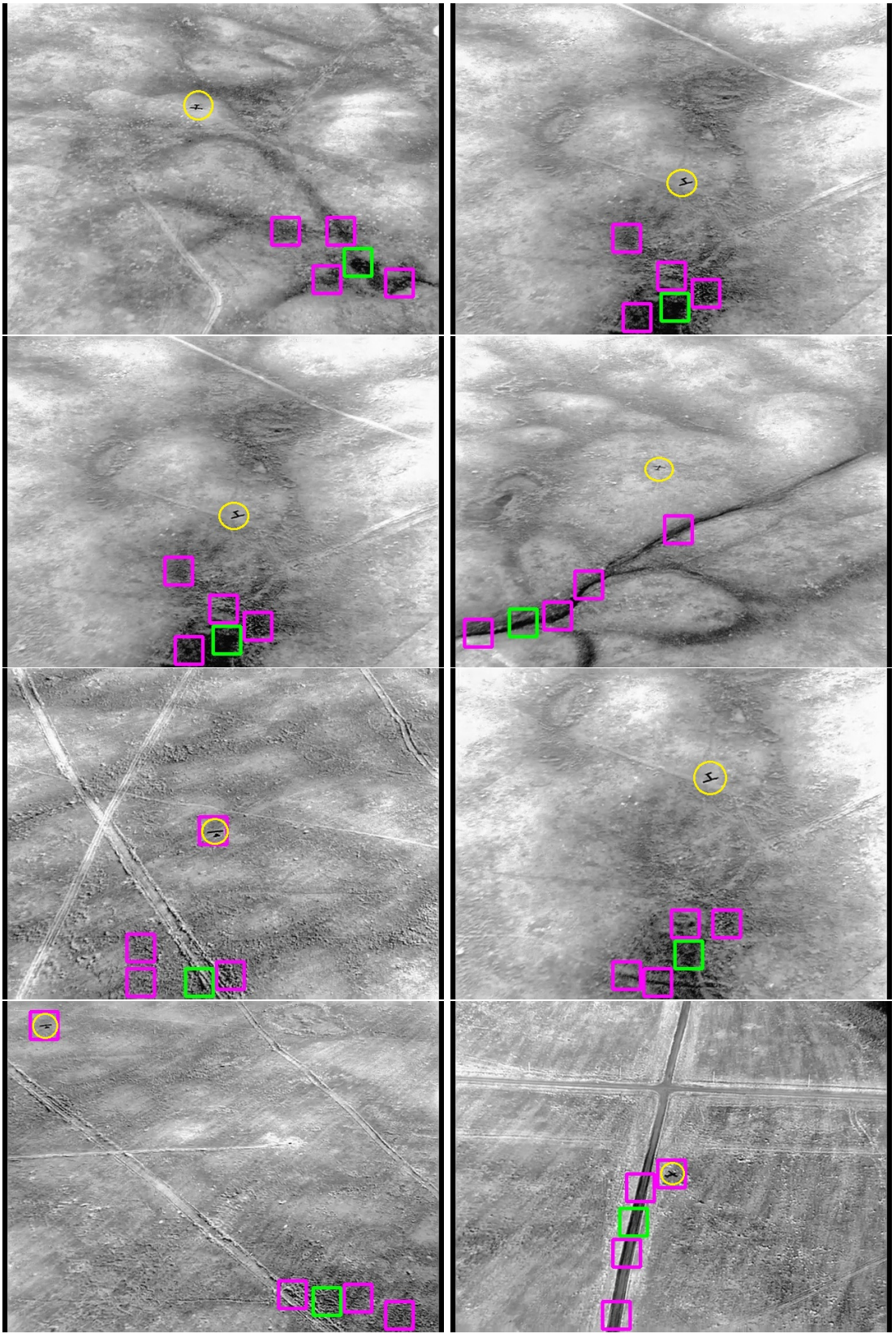

FIGURE 3.18: MF performance on a few images of the validation set. It shows the first highest score in green and the next four highest scores in magenta. Yellow circle indicates the target location.

3.2 Histogram of Oriented Gradients

3.2.1 Approach

In this section, I report on the use of Histogram of Oriented Gradients (HOG) as a detector. I implement this method in the same fashion as the MF method (see section 3.1.1). However, instead of taking the inner product of the template with image patches ((A) in section 3.1.1) or image ((B) in section 3.1.1), I compute the HOG features for each image patch or image (see section 1.4). I then use the support vector machines (SVM) module implemented in Python to classify the images from the HOG features.

Each image is broken up into blocks and each block into cells. I chose 2×2 cells per each block. The number of the unsigned directions from 0 to 180 degrees is given by the parameter *orientations*. Table 3.1 shows the HOG parameter values I used for each method of sliding window. I chose these parameters by determining the values that maximized the accuracy of the SVM classifier when tested on the validation set (described in section 2.6). For example, Table 3.2 shows the accuracy of Geometry method with 2×2 cells per block and for different values of pixels per cell and orientations. We can see that the cell size of 10×10 pixels with 7 unsigned orientations results in the greatest accuracy.

Since contrast is normalized for each block (the *block_norm* parameter above), I worried this might affect the results. For example, if the contrast is much higher in one block compared to another, then the gradient magnitude might end up lower (since it is normalized by the contrast in its own block). Therefore, gradient magnitudes are not comparable between blocks. I thus tried to implement HOG from scratch to have direct control over the normalization, as I could not simply turn it off in the Python implementation. While our implementation did not perform as well as the one in Python, the L1 normalization had almost no effect on performance (our implementation with and without normalization achieved an accuracy of about 98%). Hence, I conclude that using block contrast normalization for our infrared images does not likely result in worse performance.

Figure 3.19 shows a few visualizations of the HOG features in image patches that contain a target. The lines indicate the direction of the brightness gradient, while the brightness indicates the gradient value (the higher the gradient, the brighter the line). The brightest areas in the image patches correspond to the locations of the targets. Therefore, I can see that, in these examples, the targets are distinguishable from the background. This suggests that this approach may be more promising than the MF method.

The SVM was trained on whole training dataset. I used a regularization value of 1 (C in equation 1.15) (note that the strength of the regularization is inversely proportional to C) and

Method	*pixels_per_cell*	*orientations*
10×10	5	9
20×20	8	9
40×40	10	9
80×80	12	9
Geometry	10	7

TABLE 3.1: Specification of the HOG parameters in the "skimage" library used to extract features of the airplanes in our dataset.

pixels_per_cell	orientations	Accuracy
4	7	97.61
4	8	97.48
4	9	97.51
6	7	98.10
6	8	98.13
6	9	98.17
8	7	98.12
8	8	98.14
8	9	98.22
10	7	98.32
10	8	98.22
10	9	98.23
12	7	97.46
12	8	97.48
12	9	97.5
14	7	97.45
14	8	97.32
14	9	97.43
16	7	95.75
16	8	95.83
16	9	95.73
18	7	89.20
18	8	89.79
18	9	88.99

TABLE 3.2: Accuracy of the SVM classifier applied to features extracted with HOG on the test set. The accuracy depends on the parameter values chosen. Red accuracy value indicates the highest value obtained.

a linear kernel (described in section 1.4.1.5). I did not find any major qualitative differences in results by adjusting these parameters.

3.2.2 Results

As with MF, HOG (+ SVM) performs differently depending on the size of window that I use. When choosing a window size, several aspects are at play that may increase or decrease the

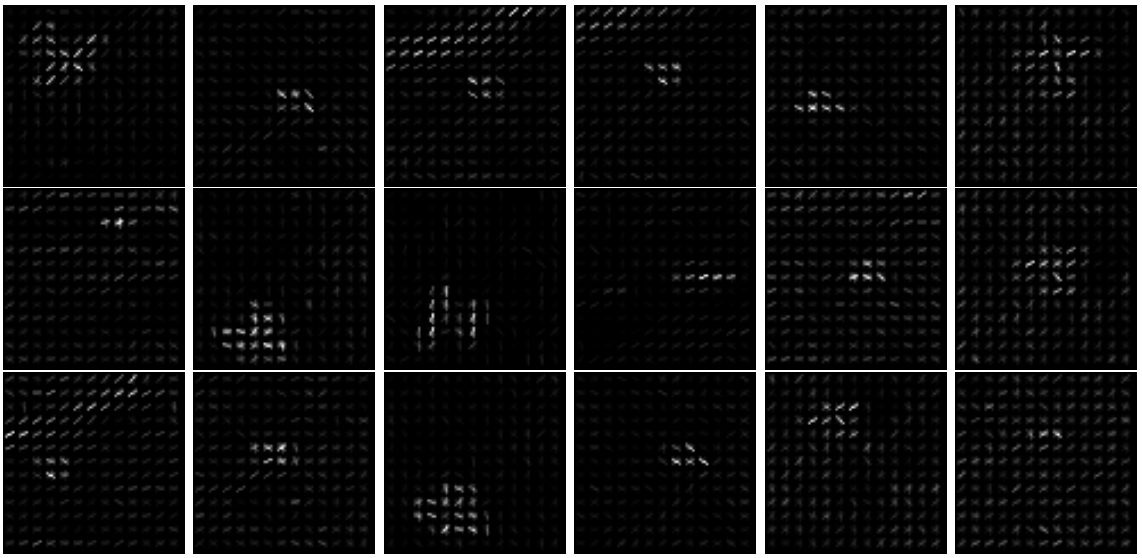

FIGURE 3.19: HOG feature visualization for image patches of 40×40 pixels. Targets correspond to the brightest areas in the patches and are therefore distinguishable from the background.

performance of the algorithm. On the one hand, having too large of a window may lower the algorithm's performance, since: (1a) most of the information in the window may be of the background rather than the airplane, so that it might be more difficult to detect the airplane; and (2a) there are many more possible positions within the window the airplane may appear in. Indeed, (1a) is the main reason to use a sliding window approach rather than to simply apply the algorithm to the entire image at once. On the other hand, a smaller window size may also lower the algorithm's performance, since: (1b) having some of the environment in the background to contrast against the target can provide helpful detection information, and too small a window size may provide too little of the environment; and (2b) there is a higher probability that the target will only be partly contained within the sliding window. Since the training set consists of whole airplanes, the algorithm will have greater difficulty in detecting partial planes. The best window size is the one that balances all of these factors within a particular dataset.

When I try different window sizes, we can see all of these factors reflected in the performance of the algorithm. First, I look at the image patches ((A) in section 3.2.1) that have a balanced number of positive and negative images. In this case, since I am not using a sliding window, all of the airplanes in the training and test sets are whole airplanes. Therefore, (2b) does not apply, and as long as enough background is present, I would expect that (1a) and (2a) would both apply, so that the algorithm would perform best with a window size based on our Geometry method and worst with our smallest window size of 10×10 pixels.

In this case, there are slightly more positive than negative images, hence when $recall = 1$, the *precision* is slightly higher than 0.5. As expected (since the dataset is balanced), the ROC curves for the image patches in Figure 3.23 reflect the same performance as in the PR

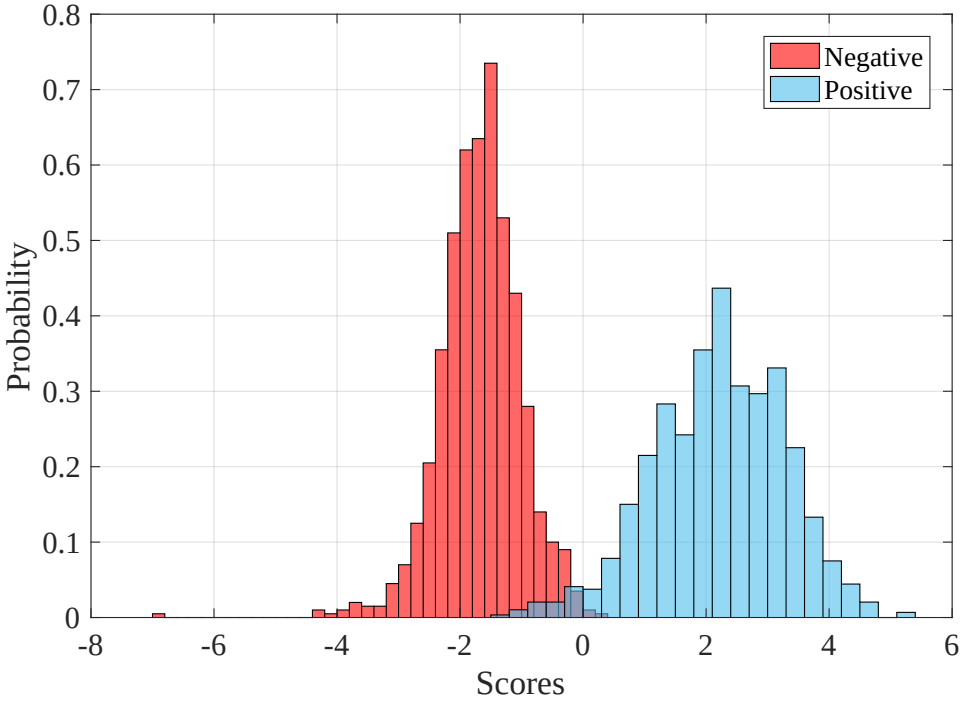

FIGURE 3.20: Probability distribution of the SVM decision value on positive and negative image patches of size 40×40 pixels based on 10,000 image patches from the training set extracted using Method (A)(Section 3.1.1).

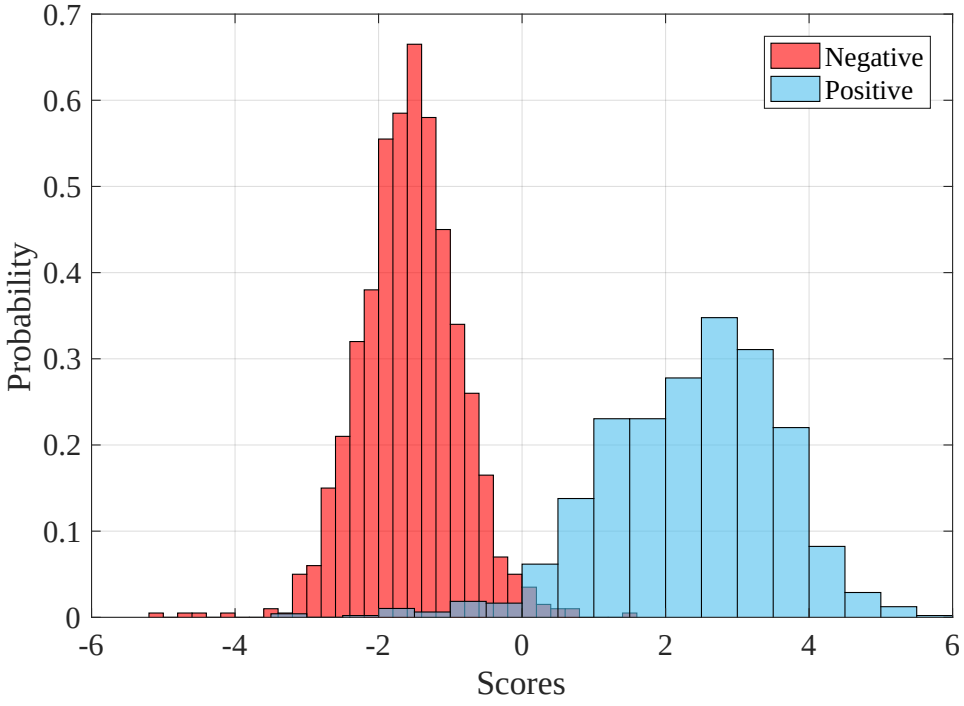

FIGURE 3.21: Probability distribution of the SVM decision value on positive and negative image patches of size 40×40 pixels based on 10,000 image patches from the patch-based validation set extracted using Method (A)(Section 3.1.1).

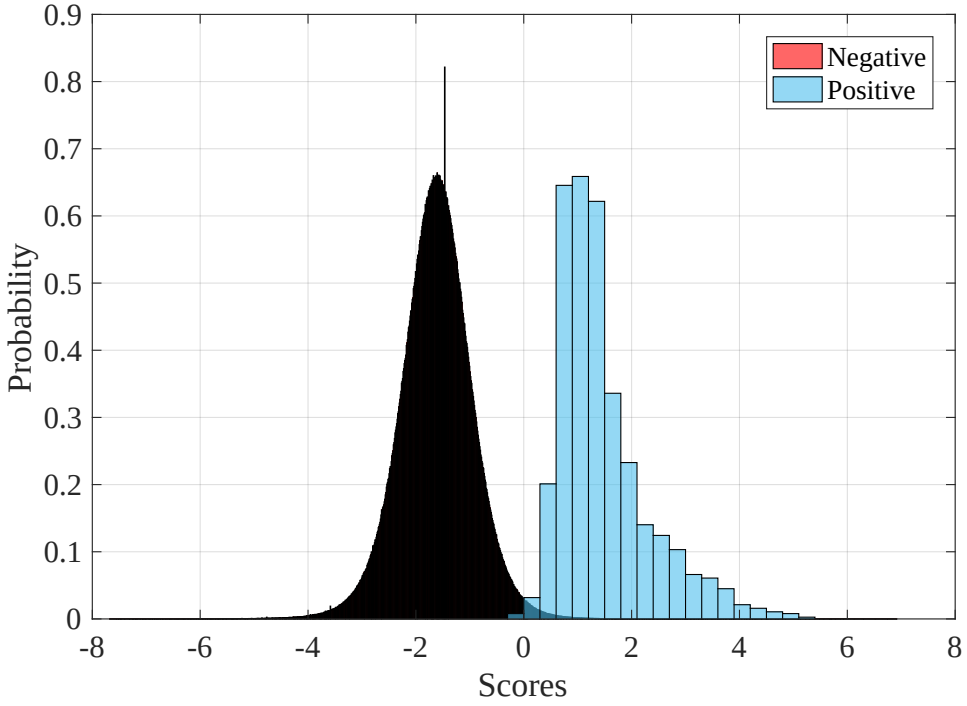

FIGURE 3.22: Probability distribution of the SVM decision value on positive and negative image patches of size 40×40 pixels based on images of size 480×640 pixels from the validation set using Method (B)(Section 3.1.1).

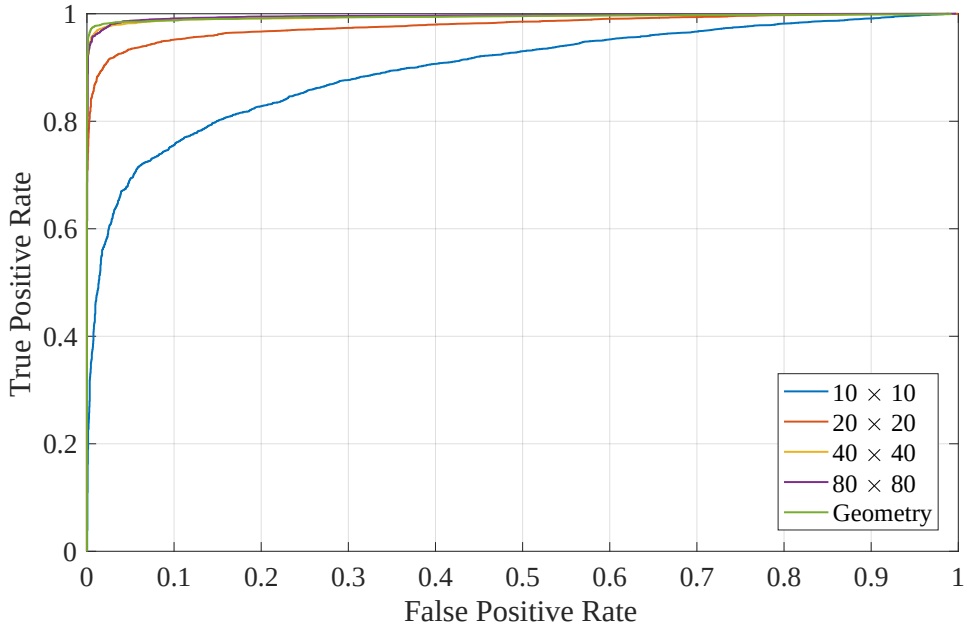

FIGURE 3.23: ROC curve of HOG+SVM with different sliding window sizes on the validation set image patches.

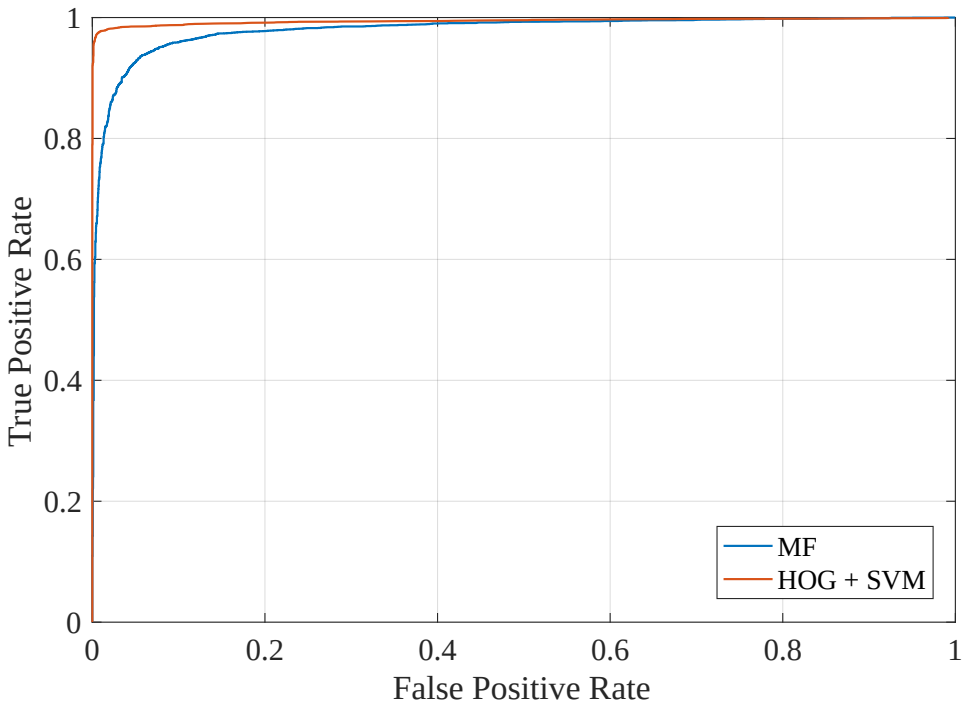

FIGURE 3.24: ROC curve of MF and HOG (+SVM) based on Geometry method on the validation set of image patches.

curves. Figure 3.24 shows MF performance on the balanced dataset next to the HOG (+SVM) performance based on the Geometry method.

In the case of implementing HOG (+ SVM) on the entire image with a sliding window ((B) in section 3.1.1), I can see in Figure 3.27 that HOG performs better than MF in the sense that there are thresholds that allow for higher, though still low, *precision*. It also appears that the Geometry method produces the best performance, and the smallest, 10×10 window size the worst. Figure 3.20 shows the probability distributions of SVM decision value for the training image patches and Figure 3.21 shows the probability distribution for the patch-based validation set extracted by method (A)(section 3.1.1). Figure 3.22 shows the same distributions for patches extracted with method (B) after implementing the non-maximum suppression.

The geometry method attains optimal performance with scaling constant $C = 34,000$ which is somewhat larger than for the MF detector. This suggests that, for HOG, having more background in the the image patch can improve the performance of the detector. Figure 3.26 shows HOG (+SVM) performance over different overlaps in non-maximum suppression for the Geometry method. I chose the overlap 0 as it has a better performance not only for the Geometry method but also with different window sizes. In Figures 3.28, 3.29, 3.30, and 3.31, I examined the most confident FN, the most confident TP, the most confident FP, and the most confident TN, respectively, to try to gain insight into where the algorithm runs into difficulties.

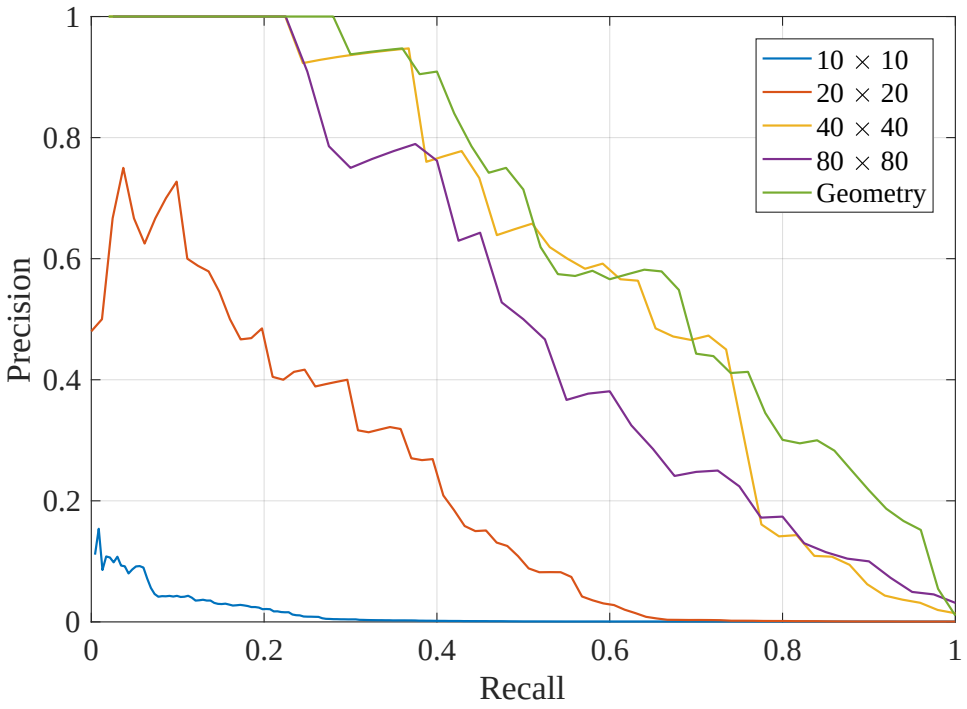

FIGURE 3.25: PR curve for the validation set of 100 images size 480×640.

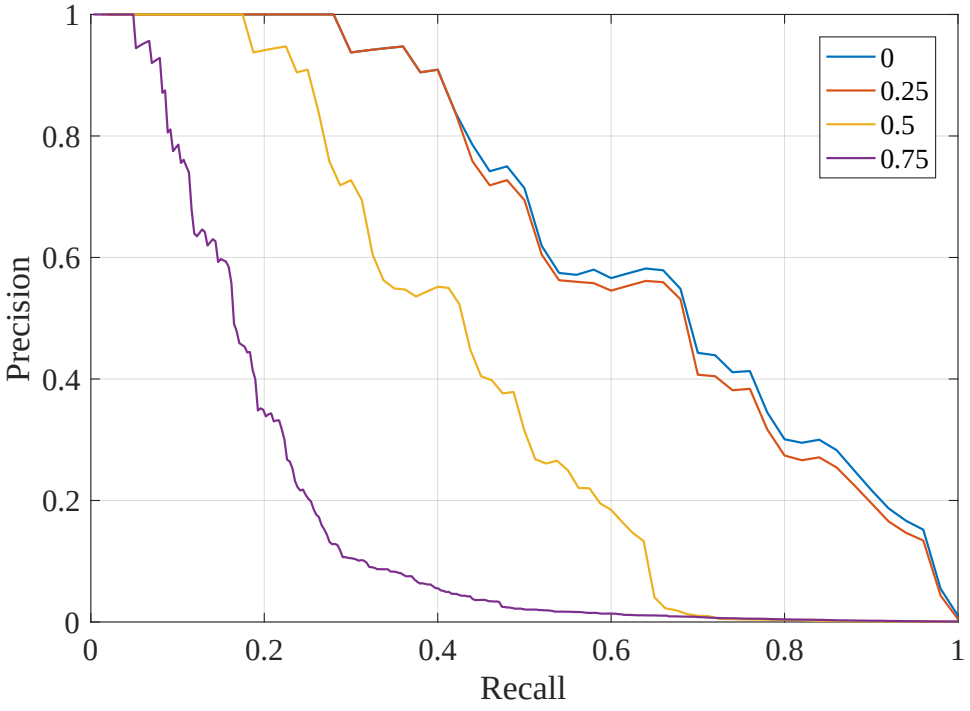

FIGURE 3.26: Precision-Recall curve of HOG(+SVM) over the IR images with different overlaps in non-maximum suppression for Geometry method.

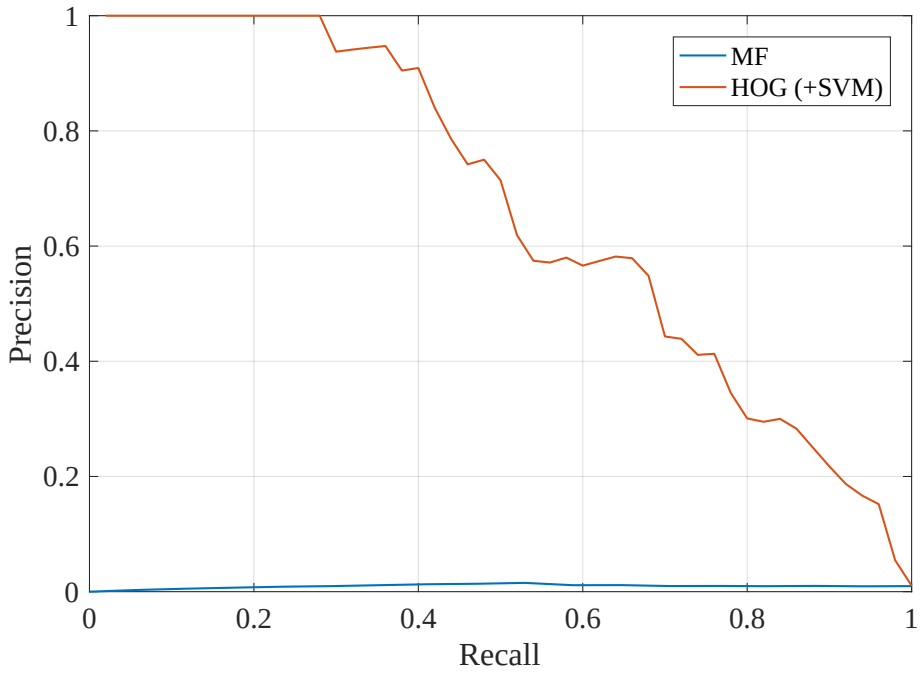

FIGURE 3.27: PR curve of MF and HOG (+SVM) based on the Geometry method on the validation set.

HOG appears especially to have difficulties in detecting small, blurry, or low contrast airplanes, as might be expected. As I again only am looking for whole airplanes within a window in the test set, I constrained the annotated target location to be in center of the window. In future work, if we randomized the target position within the windows in the training set, we expect, to achieve slightly better performance. We would then expect the SVM to learn to classify uncentred targets better. I only included centred targets in the training set, and I found that the most confident TP were image patches with centred targets, while the most confident FN were image patches with uncentred targets. The algorithm also more easily identifies backgrounds that are more spatially uniform, while having difficulty with backgrounds with strongly linear contrast changes. The algorithm false alarms if there is a clear, high-contrast, sharp contour running through the patch, near to the center.

Figure 3.32 shows the most confident detections, after non-maximum suppression, identified by HOG + SVM for some example images. For the Geometry method, it took 133.25 seconds to train the SVM and 174.32 seconds for implementing the algorithm for each validation image.

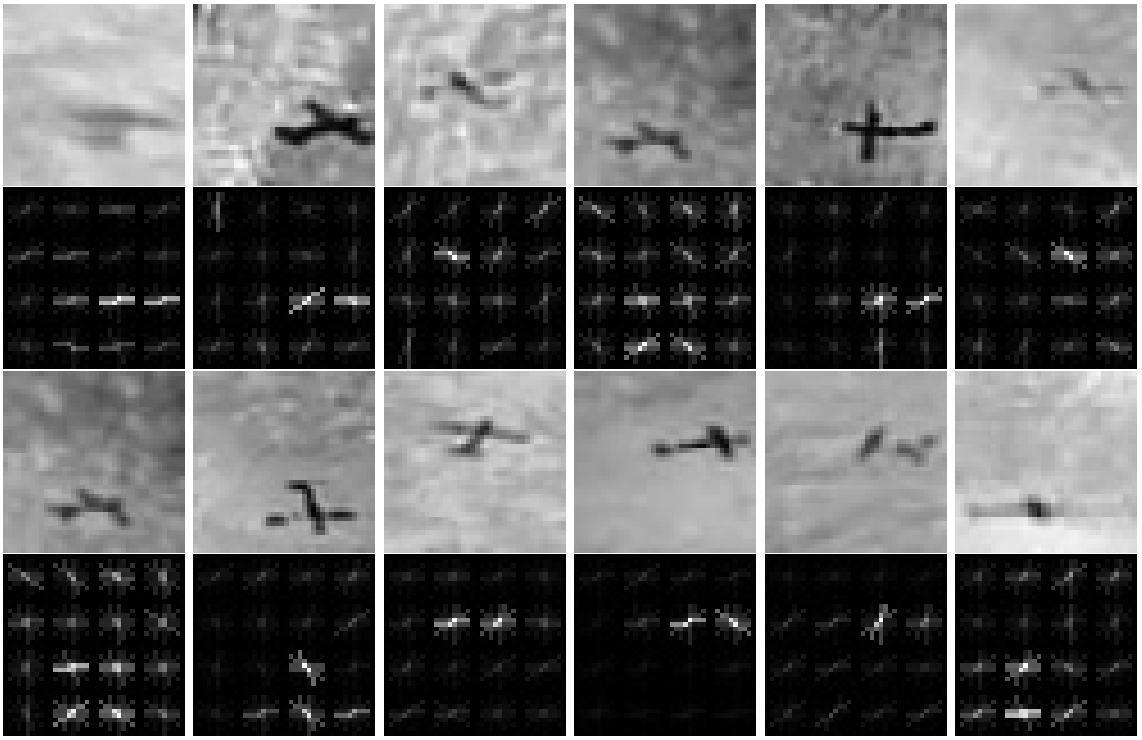

FIGURE 3.28: Most confident false negative patches of the HOG + SVM algorithm.

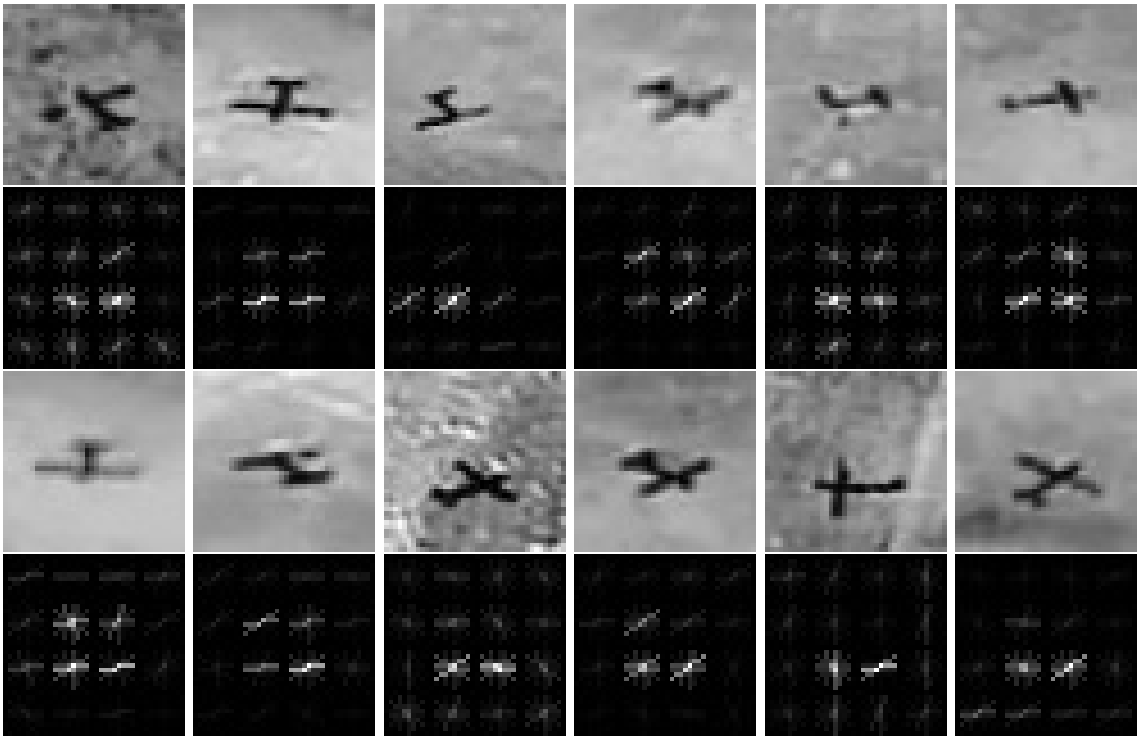

FIGURE 3.29: Most confident true positive patches of the HOG + SVM algorithm.

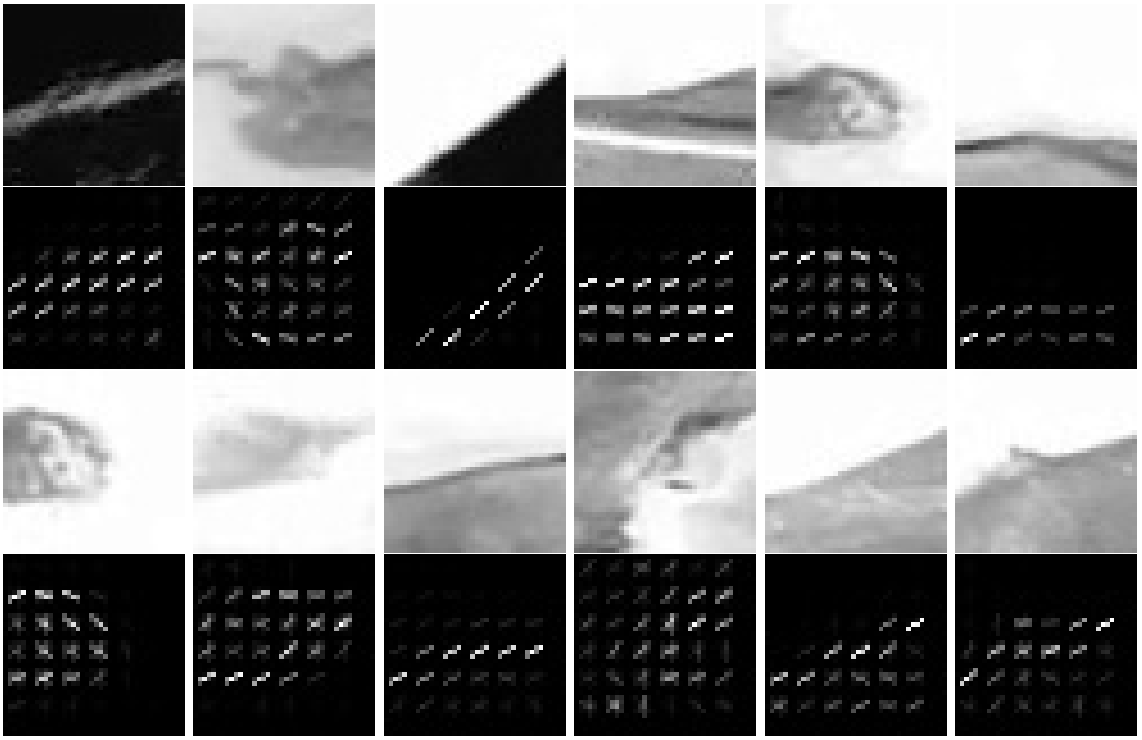

FIGURE 3.30: Most confident false positive patches with the HOG + SVM algorithm.

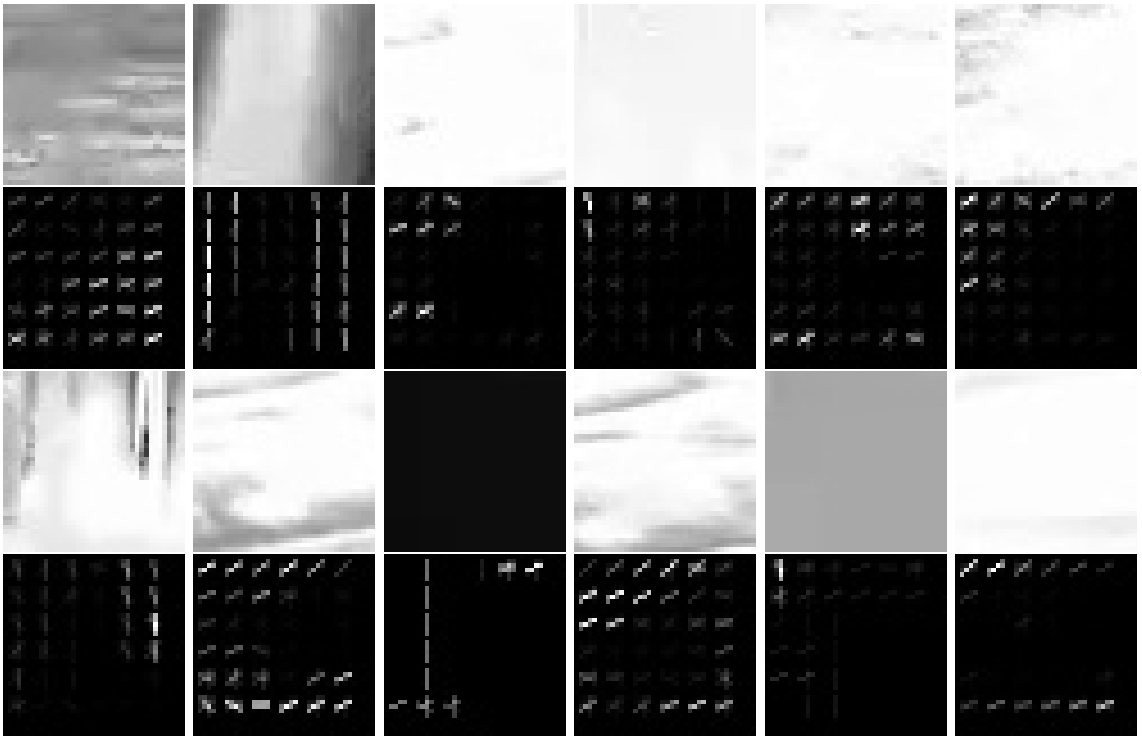

FIGURE 3.31: Most confident true negatives with the HOG + SVM algorithm.

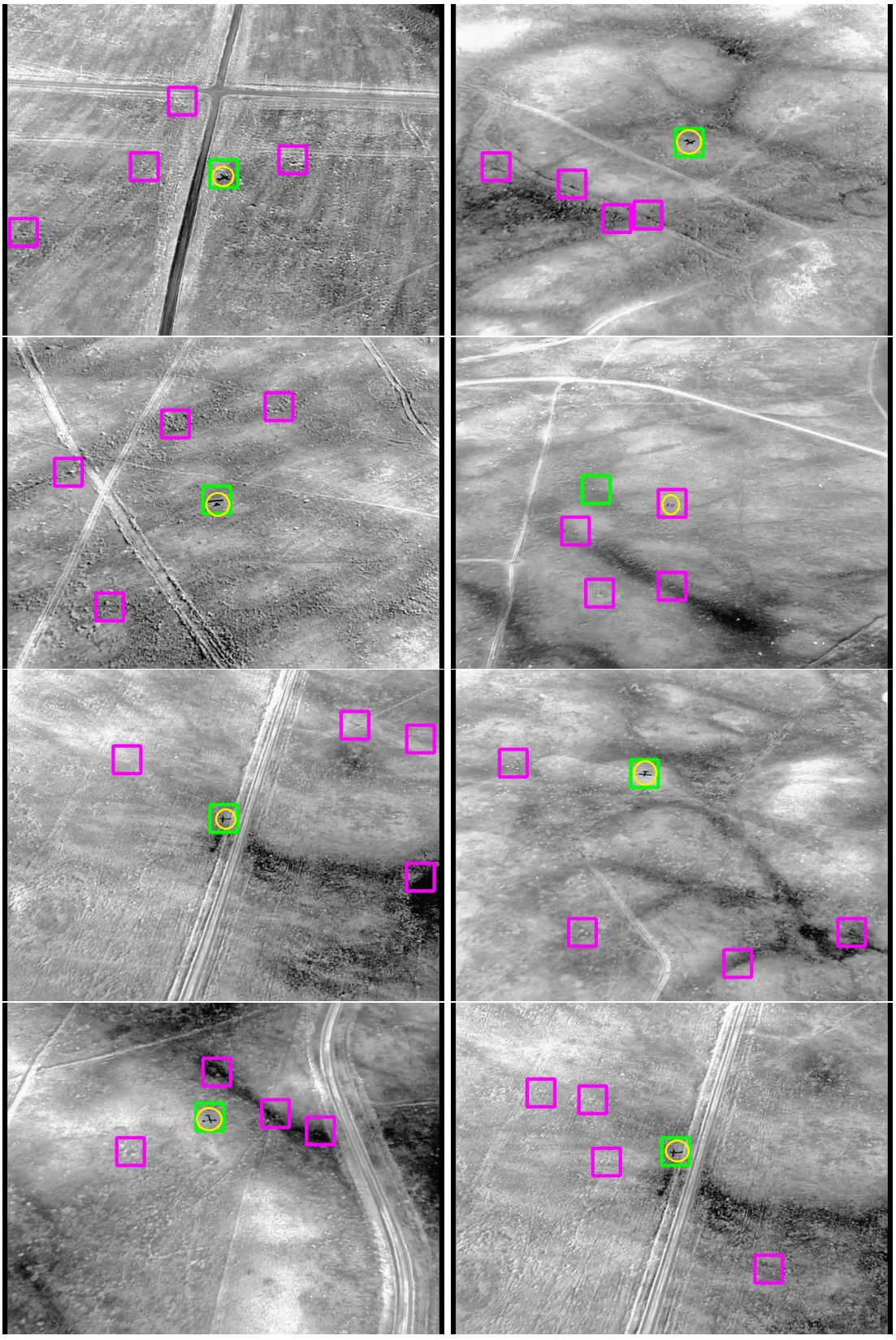

FIGURE 3.32: HOG + SVM performance on a few images. It shows the first highest score in green and the next four highest scores in magenta. Yellow circle indicates the target location.

3.3 Deep Learning

3.3.1 Approach

For deep learning, I implemented a neural network using Faster R-CNN (Region-based Convolutional Neural Network) as the model and ResNet50 (Figure 3.33) as the backbone.

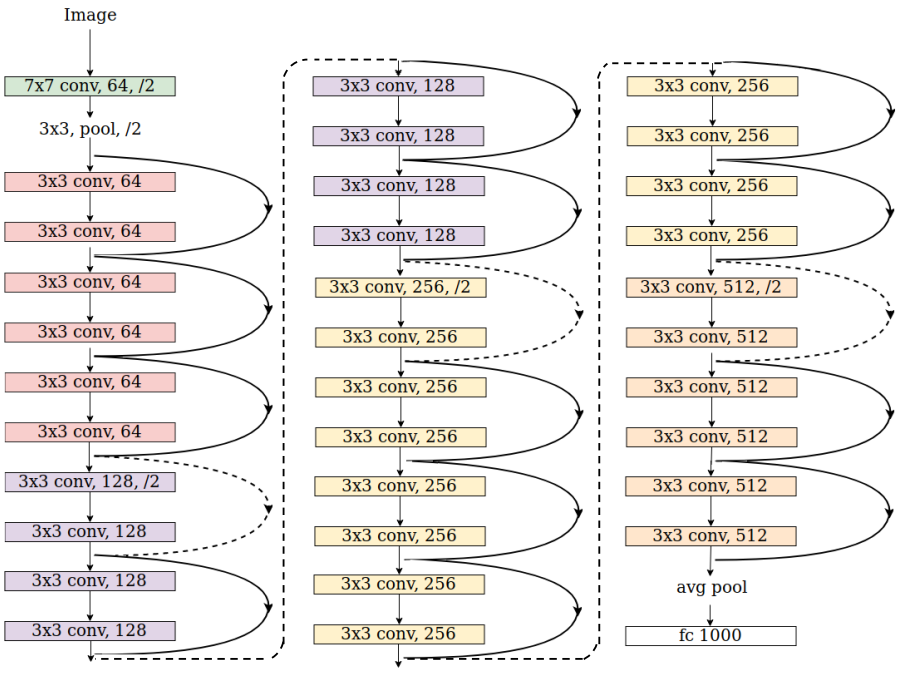

FIGURE 3.33: ResNet50 structure. It consists of 49 convolutional layers and one fully connected layer[34].

Deep residual networks were groundbreaking in the field of deep learning. ResNet makes it possible to train thousands of layers and achieve very high performance [34]. A well-known difficulty with deep networks is the vanishing gradient problem. Basically, the loss gradients, and therefore the computed changes in the weights, at the first layers of the network can become extremely small as the gradient of the loss function is propagated backwards from the later to the earlier layers. This can slow down or even stop learning in the network during training. ResNet implements one way to prevent this issue by adding skip connections. As opposed to the MF and HOG+SVM methods, Faster R-CNN does not use a fixed sliding window to scan an image for the target. Instead, it proposes bounding boxes that might contain the target. Therefore, in addition to the annotations (i.e., the labeled (x, y) coordinates of the airplanes), I need to specify ground-truth bounding boxes around the targets as well. Based on the dimension's histogram of some targets in the dataset (Figure 2.11) I chose a bounding box of 30 pixels as the ground-truth bounding box for the positive images in the training set. I implemented Faster R-CNN with Pytorch, based on the code from GitHub[2] that I modified for our project.

[2]https://github.com/potterhsu/easy-faster-rcnn.pytorch

3.3.2 Results

I tried two different initializations for the Faster R-CNN weights: 1) Pre-trained on the ImageNet and 2) Random. I then trained both networks on our dataset for a few days, for 80,000 epochs. Figure 3.34 shows the PR curve for both initializations. I used the same training, test, and validation sets on whole images for both networks. I do not report an ROC curve on the image patches for the deep learning method since the deep learning method is based on anchors and not sliding window.

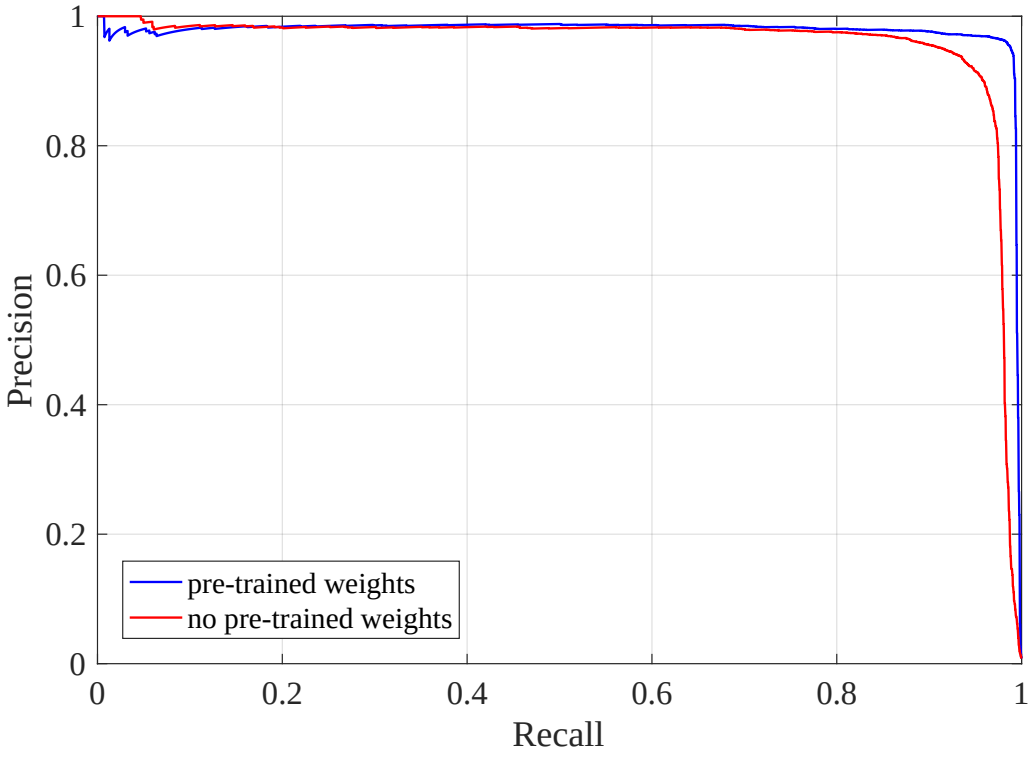

FIGURE 3.34: The PR curve of Faster R-CNN with and without pre-trained weights on whole validation set.

The training error using pre-trained weights and using no pre-trained weights reached 0.04 and 0.1, respectively. Figure 3.35 shows that Faster R-CNN performs much better than MF or HOG. Interestingly, the network that was only trained on the airplanes performs worse than the network that was pre-trained on unrelated images in ImageNet. Using the pre-trained weights might have taught the network better image statistics.

The training loss and validation loss for both cases (with and without pre-trained weights) are shown in Figures 3.36 and 3.37. After each 20 steps of training, I keep the weights fixed and compute the evaluation loss on the validation set. As shown, the training loss is higher than the validation loss at the beginning of training, but at later steps they converge. One possible reason for this is that in my Faster R-CNN implementation the hyper-parameters for regional proposal network (RPN) are different in training and validation process. More anchors are proposed in

the training process rather than in the validation process in this implementation. The more the number of proposals the bigger the loss will be, specifically at the beginning of training where network is more prone to make mistakes. The other reason that might result in higher training loss is with the current implementation each datapoint for the validation loss consists of image batch of 1, whereas it is 10 images for the training loss. We can see that the training of the Faster R-CNN with pre-trained weights is much faster than the network without pre-trained weights. For example, after 1,000 epochs, the loss for the network with random initialization is roughly 0.4, whereas for the network with ImageNet initialization, it is roughly 0.1 (Figure 3.37 and 3.36). The final accuracy at the end of training on the validation set for the Faster R-CNN with pre-trained weights is 89.68% and for the case without the pre-trained weights is 89.18%. Figures 3.38 shows some example detections from the test set. In most cases, the network is able to detect the target on the ground despite the fact that the target is very small in the image. The network can detect the target correctly even for some cases that could be difficult for a human observer.

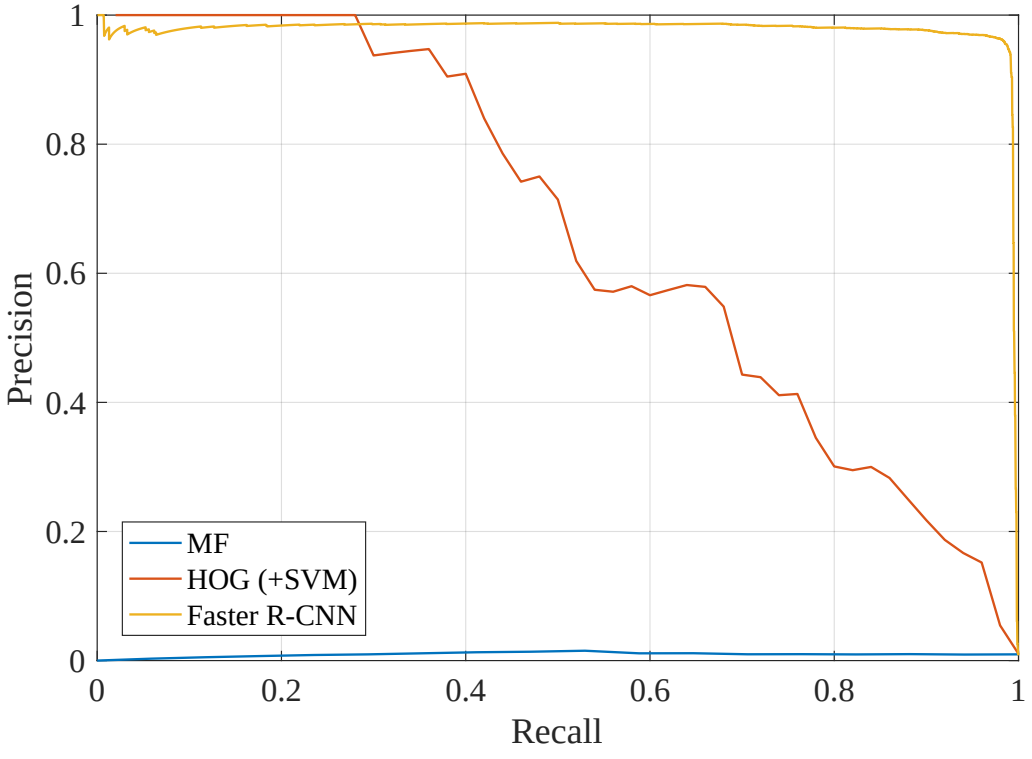

FIGURE 3.35: PR curve of MF, HOG (+SVM), and Faster R-CNN based on the validation set of images.

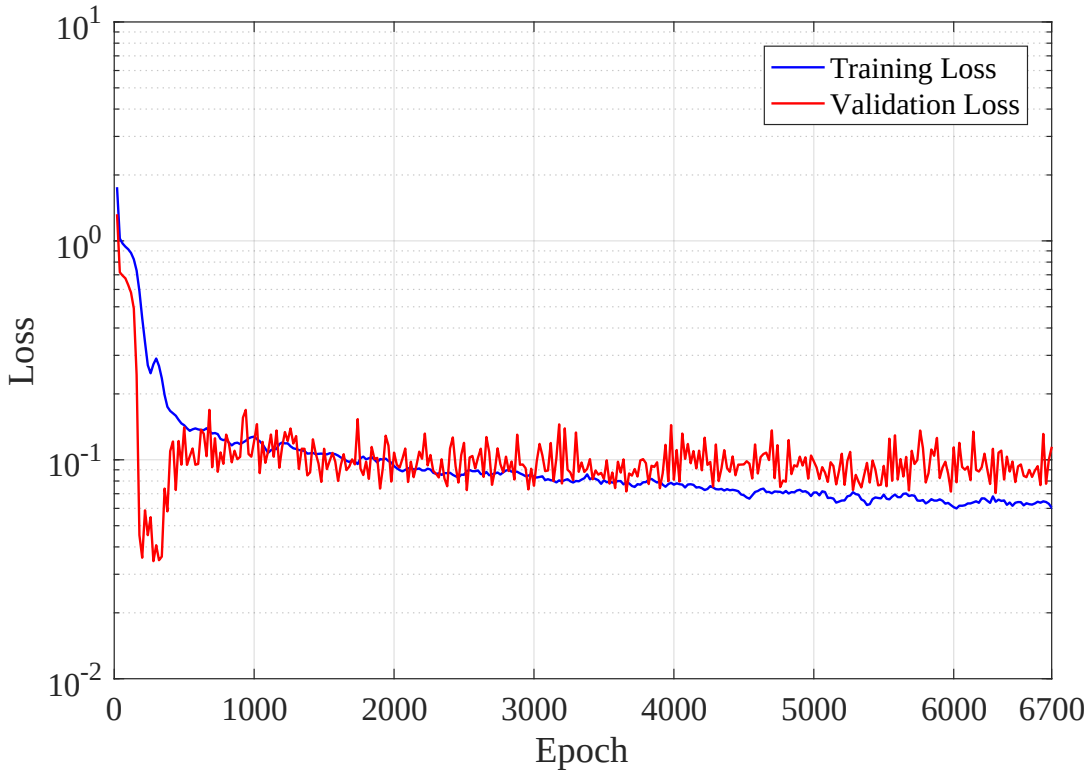

FIGURE 3.36: The training loss and validation loss of Faster R-CNN with pre-trained weights.

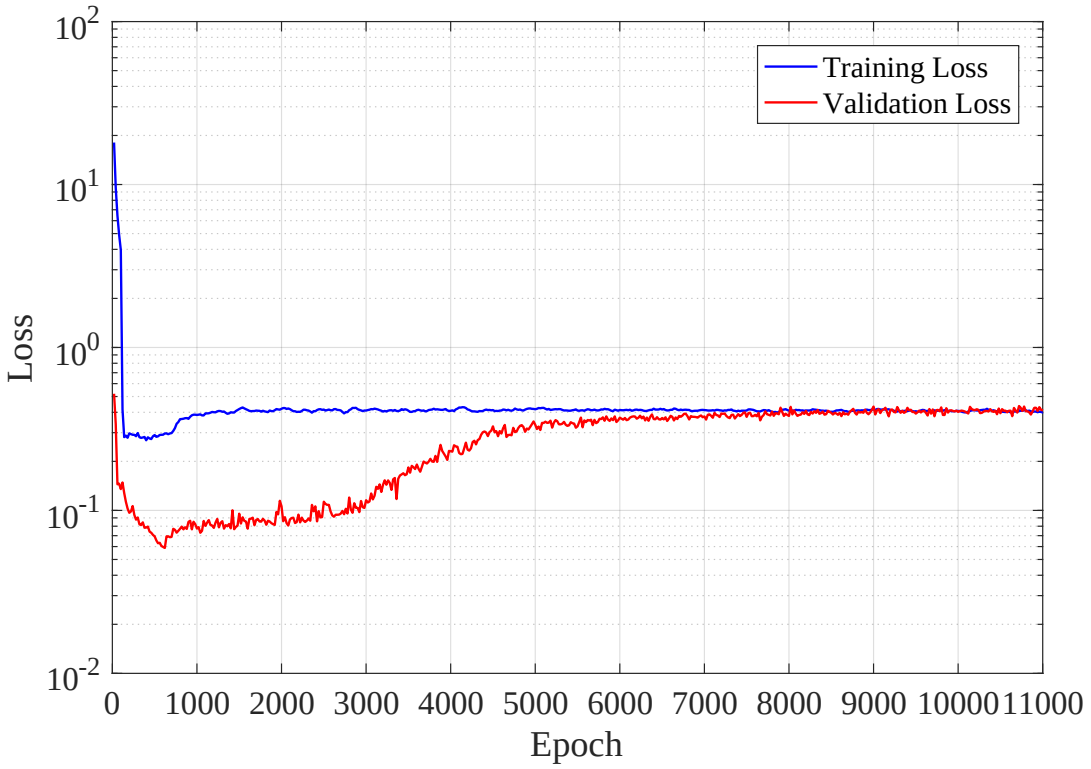

FIGURE 3.37: The training loss and validation loss of Faster R-CNN without any pre-trained weights.

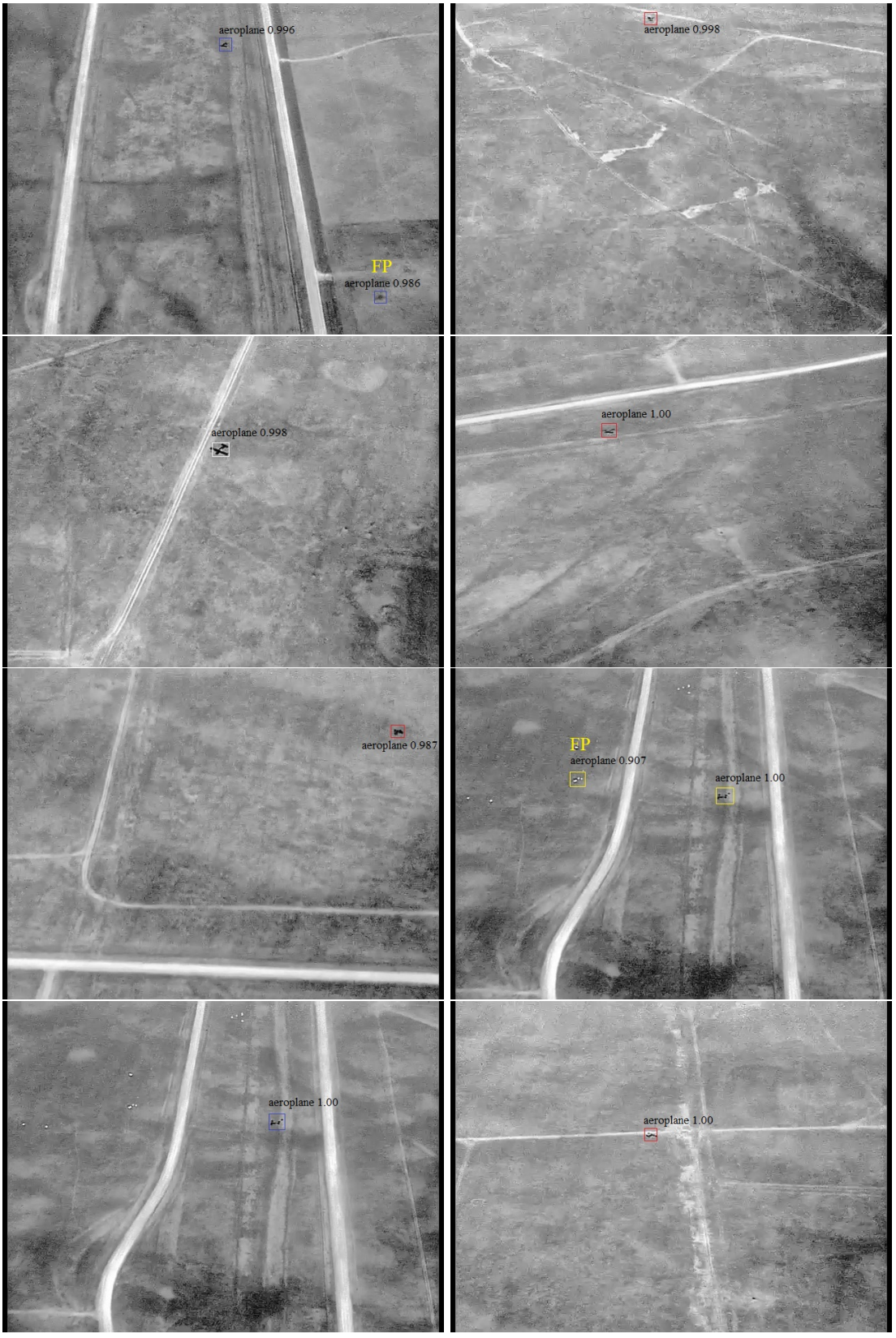

FIGURE 3.38: Example results on the test dataset for Faster R-CNN. Each image has one target. In the left column, there is one false positive in the top image and another in the image that is second from the bottom.

3.4 Discussion

I implemented three algorithms to automatically detect airplanes on the ground: MF (Matched Filters), HOG (Histogram of Oriented Gradients), and Faster R-CNN (Faster Region-based Convolutional Neural Network, a deep learning network). I found the MF method performed by far the worst of the three algorithms. The template, constructed as the difference of the average of targets and backgrounds, is unlikely to match all of the targets well. In our dataset, targets have well-defined shapes but vary in size and orientation. The best performance was achieved using the Geometry method. HOG + SVM (HOG feature vectors classified by a support vector machine) performed better than MF. I found best performance with a Geometry method that adjusts for variations in viewing distance.

For the last experiment in this project, I implemented Faster R-CNN as a detector with the Pytorch library. This deep learning method can extract more meaningful features of the object. I experimented with both ImageNet and random initialization, finding faster convergence and ultimately better performance with ImageNet-initialized weights. The network could detect airplanes with an accuracy of 89.68% with weights pre-trained on ImageNet and with an accuracy of 89.18% without the pre-trained weights. This may be because the network has kept information in its weights about image statistics that may be relevant here.

We can assume that in our performance evaluation on the test set, there are n_P positive images and n_N negative images. For a fixed threshold, let true positive rate (TPR) or recall be equal to the probability of a detection given a positive image ($p(r = 1|P)$). TPR happens if the detection overlaps a ground truth target, and thus qualifies as hit. Let the precision or positive predictive value (PPV) be $p(P|r = 1)$ and the false positive rate (FPR) be $p(r = 1|N)$. FPR denotes the number of detections not at the target location, per image. Let n_{TP} and n_{FP} represent the number of observed true and false positives, respectively. Then we have:

$$\text{PPV} = \frac{n_{TP}}{n_{TP} + n_{FP}} = \frac{n_P \times \text{TPR}}{n_P \times \text{TPR} + (n_P + n_N) \times \text{FPR}} \tag{3.7}$$

or

$$\text{PPV} = \frac{\text{TPR}}{\text{TPR} + (1 + \frac{n_N}{n_P}) \times \text{FPR}} \tag{3.8}$$

Let us assume that it takes 1 hour to find the airplane with a 30 f/p video. Then, n_P might only be one, but $n_N = 60 \times 60 \times 30$. Thus under realistic operating conditions, the *presicion* or PPV is:

$$\text{PPV} = \frac{\text{TPR}}{\text{TPR} + (1 + 108,000) \times \text{FPR}} \tag{3.9}$$

For example, with TPR = 0.95 and FPR = 0.01, *precision* $\approx$ 0.0008, which is too low for the system to be useful. Figure 3.39 shows the PR curve for a realistic application. If we want a recall of 0.8, there will be 100,000 false positives for every true positive. Thus, the performance is not yet at a level that is useful operationally.

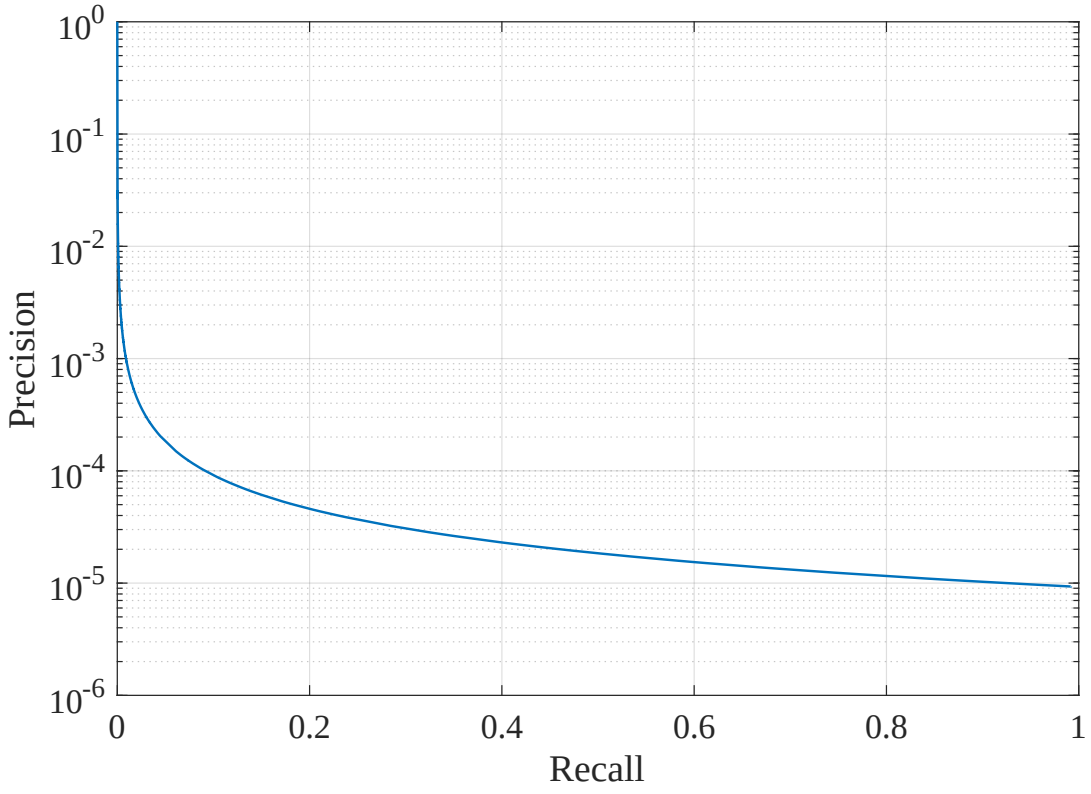

FIGURE 3.39: Precision-Recall curve of real application.

Chapter 4

Conclusion and Future Work

Our goal in this project was to implement and evaluate candidate ATD methods for finding downed aircraft in conditions that resemble those encountered in SAR (search and rescue). We used the National Research Council Flight Research Laboratory (NRC-FRL) dataset that was established in 2014 and consists of 22 hours of data with airplanes as targets. Considerable time was spent cleaning and labeling the dataset before we could apply any target detection algorithms, effort that can be taken advantage of in the future by us or others in further developing ATD technology. The three ATD methods we tested on this dataset to detect the target airplanes were MF (Matched Filter), HOG + SVM (Histogram of Oriented gradients classified by a support vector machine), and Faster R-CNN (Faster Region-based Convolutional Neural Network, a deep network learning method).

In order to train and evaluate the different ATD methods, we need to know which frames have targets, and where the targets are in each frame. This information is implicitly contained within the NRC-FRL dataset in different files that separately contain the video, information about the ground location of the targets, and information about the state of the search plane. Our first step was to put all of this information together in order to find and label each target. We first constructed an algorithm to automatically identify frames that had targets, using log files that accompanied the dataset. We then manually labeled these approximately 40,000 frames, taking care to eliminate frames that occurred during sudden camera transitions, such as rapid panning or zooming. We also eliminated zoomed-in images, since these were cases where the human operators had already located the target. Although we attempted to clean the dataset as described above, the data in the log files used to construct the algorithms were not perfectly accurate.

Among the three methods, we obtained the best performance with Faster R-CNN. MF did not perform well because it fails to capture the rotation-invariant shape of the aircraft. This can be improved by using a combination of different filters which is trained on different airplane's

rotations. While HOG (+ SVM) performed better than MF, the *precision* was still quite low. MF and HOG performed much better with Geometry method's window sizes in which the target took up enough proportion of the window area. Faster R-CNN performed better and required less training time when the network's weights were pre-trained on ImageNet than when they were randomly initialized, even though the two networks used the same training, validation, and test sets. As we outlined in section 3.4, while an ideal SAR-ATD would be able to achieve extremely high *precision* and *recall* simultaneously, we may be able to sacrifice one for the other. In SAR, we want to ensure we don't miss the target. However, a human operator might be able to screen positive detections, so that apparently low *precision* levels might be acceptable in this situation. The performance achieved here by Faster R-CNN, might not be sufficient for SAR-ATD applications as described in section 3.4. However, we believe we can further improve the performance to higher levels with some additional modifications. For example, we can use the hard-negative mining method by adding high-confidence false positives to the training set to improve the performance of the algorithm.

Another opportunity to improve performance is to take advantage of information present in nearby video frames. Consecutive frames have high spatial correlations. If a target is detected in one frame, it should also be detected in nearby frames. Combining detector outputs over successive frames can thus be used to improve the detector sensitivity. However, since the aircraft is moving, combining the detections accurately requires that we estimate the spatial mappings (roughly homographies) of a target detected in a particular frame to neighboring frames. To accomplish this, we can apply algorithms such as FAST [35], SIFT [36], SURF [37], and ORB [38] to identify features in each frame, compute pairwise matchings of the features between frame pairs, and then estimate the homography that minimizes the re-projection error. One of the limitations of the current developed assisted target detection algorithms including MF, HOG+SVM, and Faster R-CNN is that they may not be able to detect downed aircraft that have been disfigured in a crash. This must be addressed for an ATD algorithm to be of practical use.

Further improvements in dataset includes having more varied and realistic representations of target airplanes and environments that better match what might be encountered in real SAR missions. For example, we could augment and expand the current dataset to more realistically model downed aircraft by occluding parts of the airplanes with patches of background that match the image background. The NRC dataset is collected in a flat environment in the fall season. As a result, the performance of the detectors such as our deep learning method will be negatively affected in different environments, including forest, mountains, and in different seasons. To address this, we need an expanded dataset with greater variations in terrain and season.

Bibliography

[1] W. Xu, Sheng Z., Luxin Y., Fengyang W., and Weijun Z. Moving object detection in aerial infrared images with registration accuracy prediction and feature points selection. *Infrared Physics and Technology*, 92:318–326, 2018.

[2] Craig G., LeBlanc G., Thomas R., Lee M., Vinnikov M., Soffer R., Beatty H.W., and Keillor J. Flight test and statistical analysis of EO/IR sensor performance for search and rescue concept of operations development. *Technical Report LTR-FRL2016-0031, National Research Council Canada*, 2016.

[3] Xia G., Xiang B., Jian D., Zhen Z., Serge B., Jiebo L., Mihai D., Marcello P., and Liangpei Z. Dota: A large-scale dataset for object detection in aerial images. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3974–3983, 2018.

[4] Narayanan P., Christoph B., Hyungtae L., Heesung K., and Raghuveer R. A realtime object detection framework for aerial imagery using deep neural networks and synthetic training images. *In Signal Processing, Sensor/Information Fusion, and Target Recognition XXVII, International Society for Optics and Photonics*, 10646:1064614, 2018.

[5] Joseph R., Divvala S., Girshick R., and Farhadi A. You only look once: Unified, real-time object detection. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[6] Bejiga M. B., Zeggada A., Nouffidj A., and Melgani F. A convolutional neural network approach for assisting avalanche search and rescue operations with uav imagery. *Remote Sensing*, 9:100, 2017.

[7] Stone K. and Keller J.M. Convolutional neural network approach for buried target recognition in fl-lwir imagery. *In Proceeding of SPIE, Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XIX*, 9072, 2014.

[8] Olmeda D., Premebida C., Nunes U., Armingol J.M., and de la Escalera A. Pedestrian detection in far infrared images. *Integrated Computer-Aided Engineering*, 20: 347–360, 2013.

[9] Herrmann C., Muller T., Willersinn D., and Beyerer J. Real-time person detection in low resolution thermal infrared imagery with mser and cnns. *In Proceeding of SPIE, Electro-Optical and Infrared Systems: Technology and Applications XIII*, 9987, 2016.

[10] H.J. Elder, Hou Y., V. Magdin, and J. Keillor. Real-time surveillance contract no. w7711-078119001tor. *Technical Report Call-Up No.1, Defence R&D Canada*, 2008.

[11] Kim S., Woo-Jin S., and So-Hyun K. Infrared variation optimized deep convolutional neural network for robust automatic ground target recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, 2017.

[12] Wei Y., You X., and Li H. Multiscale patch-based contrast measure for small infrared target detection. *Pattern Recognition*, 58:216–226, 2016.

[13] Turin G. An introduction to matched filters. *IRE Transactions on Information Theory*, 3:311–329, 1960.

[14] Dalal N. and Triggs B. Histograms of oriented gradients for human detection. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1:886–893, 2005.

[15] Kecman V. Suppport vector machine-an introduction. *Support vector machines: theory and application*, pages 1–47, 2005.

[16] Ben-Hur A. and Weston J. A user's guide to support vector machines. *Data Mining Techniques for the Life Scieneces, Human Press*, pages 223–239, 2010.

[17] Guo Y., Yu L., Ard O., Songyang L., Song W., and Michael S.L. Deep learning for visual understanding: A review. *Neurocomputing 187*, pages 27–48, 2016.

[18] Rawat W. and Wang Z. Deep convoutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29(9):2352–2449, 2017.

[19] Lecun Y., Bengio Y., and Hinton G. Deep convoutional neural networks for image classification: A comprehensive review. *Nature*, 521(7553):436, 2015.

[20] Lecun Y., Botto L., Begio Y., and Haffner P. radient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Nair V. and Hinton G. Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[22] Yu D., Wang H., Chen P., and Wei Z. Mixed pooling for convolutional neural networks. *International Conference on Rough Sets and Knowledge Technology, Springer*, pages 364–375, 2014.

[23] Girshick R., Jeff D., Trevor D., and Jitendra M. Rich feature hierarchies for accurate object detection and semantic segmentation. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[24] Krizhevsky A., Sutskever I., and Hinton G. Imagenet classification with deep convolutional neural networks. *Advances In Neural Information Processing Systems*, pages 1097–1105, 2012.

[25] Girshick R. Fast r-cnn. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[26] Shaoqing R., He K., Girshick R., and Sun J. Faster r-cnn: Towards real-time object detection with region proposal networks. *In Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[27] Simonyan K., Vedaldi A., and Zisserman A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[28] Zeiler M. and Fergus R. Visualizing and understanding convolutional networks. *European Conference on Computer Vision*, Springer, pages 818–833, 2014.

[29] Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erthan D., Vanhoucke V., and Rabinovich A. Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[30] Xu B., Wang N., Chen T., and Li M. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[31] Sermanet P., Eigen D., Zhang X., Mathieu M., Fergus R., and LeCun Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[32] Uijlings J., Van DE Sande K.E., Gevers T., Smeulders A.W., Fergus R., and LeCun Y. Selective search for object recognition. *International Journal of Computer Vision, Springer*, 104(2):154–171, 2013.

[33] Szeliski R. Computer vision: algorithms and applications. *Springer Science and Business Media*, 2010.

[34] He K., Zhang X., Ren S., and Sun J. Deep residual learning for image recognition. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[35] Rosten E. and Drummond T. Fusing points and lines for high performance tracking. *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, 2:1508–1515, 2005.

[36] Lowe D. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision, Springer*, 60(2):91–110, 2004.

[37] Bay H., Tuytelaars T., and Van Gool L. Surf: Speeded up robust features. *European Conference on Computer Vision, Springer*, pages 404–417, 2006.

[38] Rublee E., Rabaud V., Konolige K., and Bradski G. Orb: An efficient alternative to sift or surf. *International Conference on Computer Vision*, pages 2564–2571, 2011.