Spring 5-22-2020

Ship Detection Feature Analysis in Optical Satellite Imagery through Machine Learning Applications

Sylvia Charchut
University of New Orleans, New Orleans, sylviacharchut@gmail.com

Ship Detection Feature Analysis in Optical Satellite Imagery through Machine Learning Applications

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

by

Sylvia Charchut

B.S. Southeastern Louisiana University, 2013

May, 2020

Acknowledgments

I would like to sincerely thank my supervisors Dr. Md Tamjidul Hoque and Dr. Elias Ioup, for their support and guidance throughout this research.

I would also like to acknowledge and thank my committee members, Dr. Minhaz Zibran and Dr. Mahdi Abdelguerfi, for their time and support in my thesis defense.

Table of Contents

List of Figures

List of Tables

Abstract

Ship detection remains an important challenge within the government and the commercial industry. Current research has focused on deep learning and has found high success with large labeled datasets. However, deep learning becomes insufficient for limited datasets as well as when explainability is required. There exist scenarios in which explainability and human-in-the-loop processing are needed, such as in naval applications. In these scenarios, handcrafted features and traditional classification algorithms can be useful. This research aims at analyzing multiple textures and statistical features on a small optical satellite imagery dataset. The feature analysis consists of Haar-like features, Haralick features, Hu moments, Histogram of Oriented Gradients, grayscale intensity histograms, and Local Binary Patterns. Feature performance is measured using 8 different classification algorithms, including K-Nearest Neighbors, Logistic Regression, Gradient Boosting, Extreme Gradient Boosting, Support Vector Machine, Random Decision Forest, Extremely Randomized Trees, and Bagging. The features are analyzed individually and in different combinations. Individual feature analysis results found Haralick features achieved a precision of 92.2% and were computationally efficient. The best combination of features was Haralick features paired with Histogram of Oriented Gradients and grayscale intensity histograms. This combination achieved a precision score of 96.18% and an F1 score of 94.23%.

KEYWORDS: Ship Detection, Haralick, Histogram of Oriented Gradients, Haar-like.

1. Introduction

Ship detection through satellite imagery has been a major topic of interest in both commercial and government domains for reasons such as surveillance, navigation, tourism, and trade [1]. As the number of satellite sensors has increased and, consequently, the amount of satellite imagery, so has the research on ship detection. Moreover, ship detection has proven to be a very difficult task for reasons, including image resolution issues, scene complexity, scene clutter, a lack of labeled data, and weather obstruction [2]. There are two main types of satellite imagery used for ship detection: optical and synthetic aperture radar (SAR). Unlike optical imagery, SAR imagery is not affected by weather or illumination, which makes SAR more appealing to the ship detection community. However, SAR imagery can be tainted by noise, sensitive to sea state, and experience ship reflectance issues due to ship material [3]. SAR imagery is also limited by being inherently lower resolution, less interpretable by the human eye, and experiences long revisit times compared to optical satellite imagery. Although optical imagery is affected by weather, it generally provides greater resolution and higher levels of detail [2]. Optical imagery is also more abundant than SAR imagery. Therefore, the rest of this paper will focus on ship detection in optical satellite imagery.

Ship or vessel detection research has been ongoing since the late 1970s, but due to problem complexity such as scene complexity and weather obstructions, generic ship detection algorithms do not exist [3]. Researchers are continuously improving upon ship detection algorithms with new features or deep learning [4]–[6], but these improvements fail to focus on explainability or

real-time processing. Most ship detection algorithms focus on specific environmental conditions such as nearshore detection [7-9], or open ocean [3-4], and therefore perform poorly when given a new environment. Another approach with current algorithms is the use of image processing or pre-processing techniques in order to extract ship candidates [5,9-10]. The extra steps and manual processing make these techniques unrealistic when presented with new imagery or a real-time detection scenario. Lastly, most algorithms to date employ some sort of deep neural network or a combination of classification algorithms and deep machine learning [5,11-14]. In deep learning, the algorithms require an abundance of data or data augmentation, and the architectures act as a black box [15]. Since deep learning consists of multiple layers and, consequently, millions of parameters, the ability for humans to trace or recalculate is unachievable [15]. Moreover, through the use of adversarial networks, the ability to fool a deep network with small input changes causes these networks to collapse [15]. The large data requirement becomes unsuitable for applications in which datasets are limited, and the black box architecture fails when critical tasks require explainability and resilience.

2. Literature Review

Traditionally, model development for object detection has been divided into two phases. First, extracting candidate regions then performing classification using hand-crafted features for machine learning. Therefore the community has introduced ample features and classification techniques but has not studied these features in combination or in regards to real-time

processing. The following discusses the different handcrafted features used for ship detection. Further discussion on specific features used in our research can be found in section 3.3.

Pietikäinen introduced new areas of image analysis using Local Binary Pattern (LBP) features as LBPs are robust against variations in illumination as well as being computationally efficient [16]. LBPs work by replacing pixels in a neighborhood with a binary number (0,1), based on a comparison of the neighboring pixel intensity with the center pixel intensity. Arguedas developed a vessel classifier using LBPs in optical satellite imagery, which reached an accuracy of 85.64% [17]. As an extension to LBPs, Zhu *et al.* focused on ship candidate detection in sea regions using Local Multiple Patterns (LMP) [18]. LMP extends LBP by replacing neighboring pixels with an integer value instead of a binary value, thus becoming robust against flat image regions or noisy regions. Using Support Vector Machines (SVM) for binary classification, Zhu *et al.* reached a ship detection accuracy of 92.1% [18]. Antelo, J., *et al.* use Hu moment invariants as a statistical feature for ship classification [19]. Hu moments are common statistical features used in image processing and are an extension of image moment invariants. S. Qi *et al.* introduced saliency for object detection [20], which they later improved upon with a modified Histogram of Oriented Gradients (HOG) feature descriptor called Ship Histogram of Oriented Gradients (S-HOG) [4]. HOG features are known for characterizing local shape and gradients in images and therefore are a useful feature in ship detection. S. Qi *et al*. achieved 82.8% ship detection precision for cluttered environments and 93.4% precision in quiet environments [4]. G. Yang *et al*. introduced sea surface analysis in open ocean environments, where ships are described as anomalies, and thus classification is done through anomaly detection [3]. Their sea surface analysis was performed in

multiple environments and obtained 98.88% precision in a quiet environment, however, overall precision was 89.22%.

Previous ship detection methods can be summed up as segmentation techniques, saliency detection, and anomaly detection methods. As deep learning improved upon classification tasks, most current research moved to modifications and extensions of neural networks. The following is a highlight of current deep learning algorithms used for ship detection. R. Zhang *et al.* used line segmentation and saliency along with ship head and ship body models to detect ship candidates on a dataset of 5,720 positive samples [5]. The candidates were subsequently fed into a CNN architecture for classification and produced precision results of 95.9% for inshore ship detection. In offshore ship detection, their algorithm achieved 99.1% precision [5]. Zou and Shi [21] introduced a CNN model with a linear SVM classification called SVDNet, which was based on similarities from PCANet [22]. The CNN model extracted the features which were passed to the linear SVM classifier. SVDNet was trained on 12,030 positive samples (augmented and non-augmented) and tested on 7 images, with an average precision of 72.6%. Following PCANet, N. Wang *et al.* used PCANet and LibSVM to perform classification with anomaly detection, as a means to prescreen candidates [6]. Their research included data augmentation, which produced 8,343 positive samples and a precision rate of 85%. S. Zhang *et al.* improved upon Faster-R-CNN by modifying the network structure of VGG16 and reached a precision rate between 92.95% and 97.64% [11]. Their research also used data augmentation in order to train the CNN model and avoid overfitting. Q. Shi *et al.* used a multi-feature ensemble method with multiple CNNs to perform ship detection on three different datasets [14]. Their best accuracy was 98.75% on an

augmented dataset, while their worst accuracy was 92.97% on an augmented dataset with a reduced training set. Lastly, Y. Yu *et al.* used Haar-like features in a traditional machine learning approach along with a periphery-cropped neural network to perform detection [23]. Their periphery-cropped network produced a precision of 91% [23].

As can be seen from the literature review, results can vary between datasets. The advancement in deep learning has produced significantly better classification accuracy, but deep learning tends to perform poorly on small datasets, real-time processing, and when context changes happen. The challenge of providing explainability and adapting to different environments still remains open. This research will focus on feature analysis for a small dataset, using traditional machine learning methods such as Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression (LG), Bagging (BAG), Gradient Boosting (GB), Extreme Gradient Boosting (XGB), and Extremely Randomized Trees (ET). The rest of the paper is organized as follows: In section 3, we describe the data collection, selected features, feature extraction, and classification methods used. Section 4 covers the feature analysis and classification results. Finally, section 5 concludes the paper and discusses future work.

3. Methodology

In this section, we discuss the data collection and data processing performed. We also discuss the feature extraction for each algorithm and the machine learning methods applied.

Figure 3.1 shows a high-level view of the experiment design. The dataset is processed through each feature extraction method. The feature extraction creates a comma-separated values feature file. Each feature type and variation create a separate feature file, and thus for the 15 features and variations used in this research, there are 15 feature files. The individual feature files are then processed by the 8 classification algorithms separately, meaning a single feature file once trained through the 8 classifiers will have 8 individual classification results. All 8 classifiers were implemented using 10-fold cross-validation. The classifier training for all 15 individual feature files produces 120 (15 feature files × 8 classifiers) classification result files.

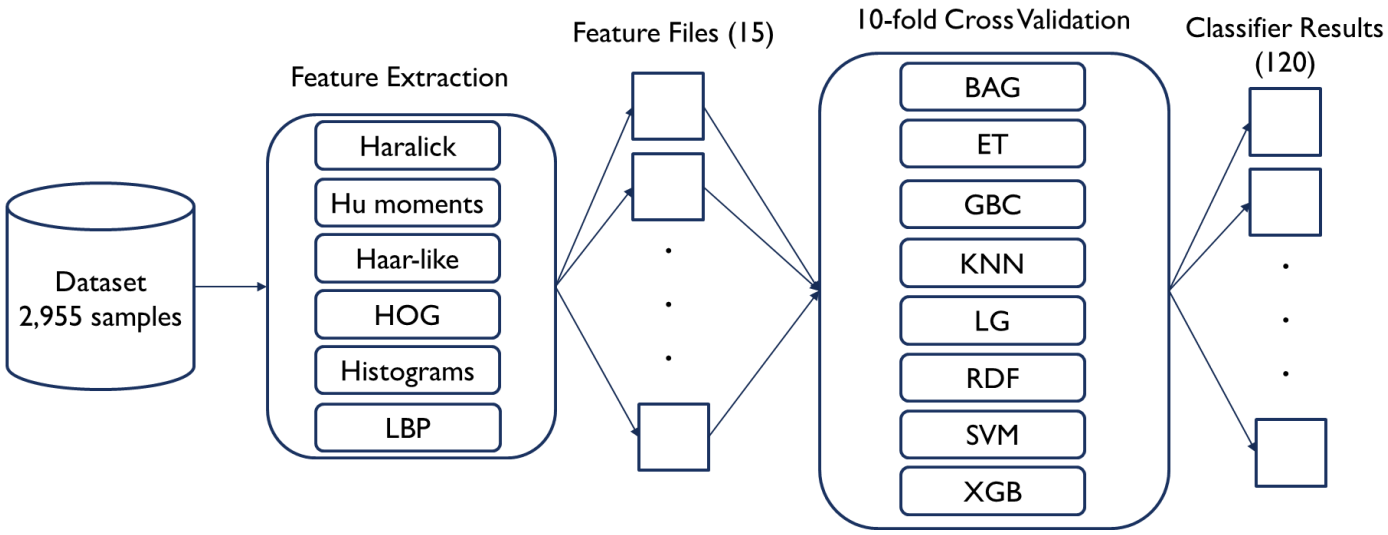


Figure 3.1: This diagram shows the experiment design and process of feature extraction to classifier training.

3.1 Data Collection

For our research, the dataset was collected from DigitalGlobe [24] satellite imagery around the Singapore Strait and manually curated. DigitalGlobe is a freely available source for satellite imagery. The Singapore Strait was chosen due to its high vessel traffic as a major port. The dataset

consisted of 100 optical satellite images ranging in size from 1033×854 pixels to 1083×7119 pixels. Each image was manually cropped into 250×250 samples which totaled to 1300 samples, 300 positive multiship samples, and 1000 negative multiship samples. The 300 positive 250250 samples containing multiple ships were then annotated using a tool called BBox-Label_Tool [25]. BBoxLabel is an open-source tool, which opens a simple GUI that allows the user to load a directory of positive images and annotate multiple objects as well as multiple classes for each image. After labeling our 300 positive samples, they were cropped into single vessel samples of size 24×24 and totaled 955 positive vessel samples. The 1000 negative samples were also divided into a total of 2,000 smaller samples of size 24×24. Our dataset contained a mix of calm and cluttered environments, including cloud coverage, docks, islands, land, contrast differences, and textured surfaces. Figure 3.2 depicts examples of 4 different positive ship samples and 4 different negative non-ship samples from the GlobalView dataset.

Figure 3.2: The top row of images represent 4 different positive ship samples in different environments. The bottom row of images represent 4 different negative non-ship samples, including land and cloud obstruction. These images represent different challenges presented in the dataset.

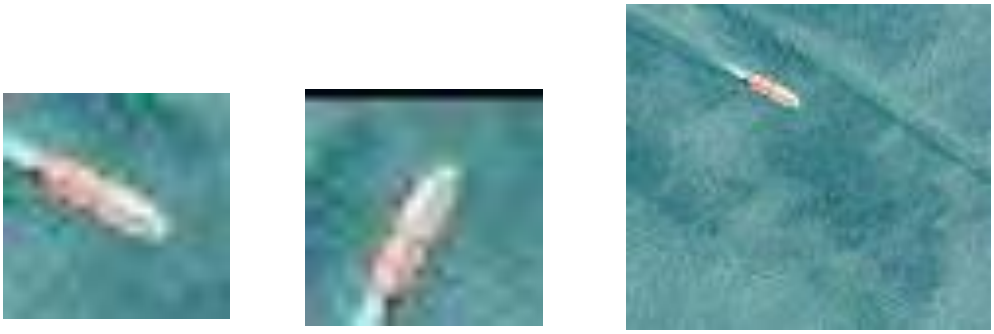
3.2 Features

The following discusses the multiple textures and statistical features used in this research. For each positive and negative image, the features extracted were Hu moments, Harlick features grayscale histograms, Histogram of Oriented Gradients, Haar-like features, and Local Binary Patterns. These features are known to extract patterns, statistical information, and in some cases, provide invariance to scale, rotation, and translation, thus containing more information than the individual pixels themselves. We discuss each feature below generally as well as specifically in regards to our dataset.

3.2.1 Hu Moments

Ming-Kuei Hu introduced moment invariants in 1962 to capture patterns in images while being invariant to image transformations [26]. Hu moments are an extension of image moments. Image moment invariants are simply a weighted average of image pixel intensities. Hu moments extend image moments in order to be invariant to scale, translation, reflection, and rotation. There are 7 Hu moments, and these moments are based on central moments. Hu introduced 6 absolute orthogonal invariants and one skew orthogonal invariant, those have proven to be adequate measures of tracing image patterns with the assumption of images being continuous functions and noise-free [27]. Of note, Huang et al. analyzed Hu moment invariants with respect to image resolution and image transformations to conclude that with adequate resolution, the moment invariants change only slightly [27]. We discuss this observation since our dataset is composed of

very small samples and thus may not have the adequate resolution. Figure 3.3 displays a positive image sample, a rotation of the positive sample and a different scaled version. Table 3.1 demonstrates the effectiveness of Hu moments on the 3 images from Figure 3.3 as the values for each of the 7 Hu moments across all 3 images are relatively close together. For each image, we generate 7 Hu features using OpenCV's moments and HuMoments functions [28].



A B C

Figure 3.3: Starting from the left is a clear positive sample (A) from the dataset, the following image (B) is image (A) rotated by 90 degrees. Lastly, image (C) is a different scaled version of image (A).

Table 3.1: The 7 Hu moments for the 3 image samples in figure 3.3. The Hu moments for the 3 images are close in value and reflect the Hu moment's invariance to scale and rotation.

Hu Moments	Figure 3.2A	Figure 3.2B	Figure 3.2C
H[1]	2.92409325	2.92403239	2.88520941

H[2]	8.9214574	9.84329869	7.70497829
H[3]	11.54743336	11.62098777	12.63939981
H[4]	12.11103466	12.04494061	12.79032651
H[5]	24.21725739	24.3125476	25.5263543
H[6]	16.57384259	-16.96660028	16.71029491
H[7]	-24.01138203	-23.90942647	-26.0212579

3.2.2 Haralick Features

R. Haralick introduced Harlick features in 1973 as a general procedure for extracting texture features during image analysis [29]. The basis of Harlick features is the gray-level co-occurrence matrix, which quantifies the spatial relationship between neighboring pixels. The co-occurrence matrix is created by counting the number of instances a pixel value of *i* is adjacent to a pixel value of *j* then dividing the entire matrix by the number of comparisons. From the co-occurrence matrix, the 13 textural properties described in [29] are statistically computed. Harlick features are also typically rotation invariant, making them particularly useful in ship detection. **Figure 3.4:** Starting from left to right, the first matrix (A) is an example of an input image consisting of 3 gray

values. The second matrix (B) is the gray-level co-occurance matrix displaying the neighboring pixel combination frequencies. The third matrix (C) is the normalization of the grey-level co-occurance matrix. The equations (D) at the end are two of the thirteen Haralick properties that are computed from the normalized co-occurance matrix (C).Figure 3.4 is an example diagram showing how the grey-level co-occurrence matrix and the Haralick properties are calculated.

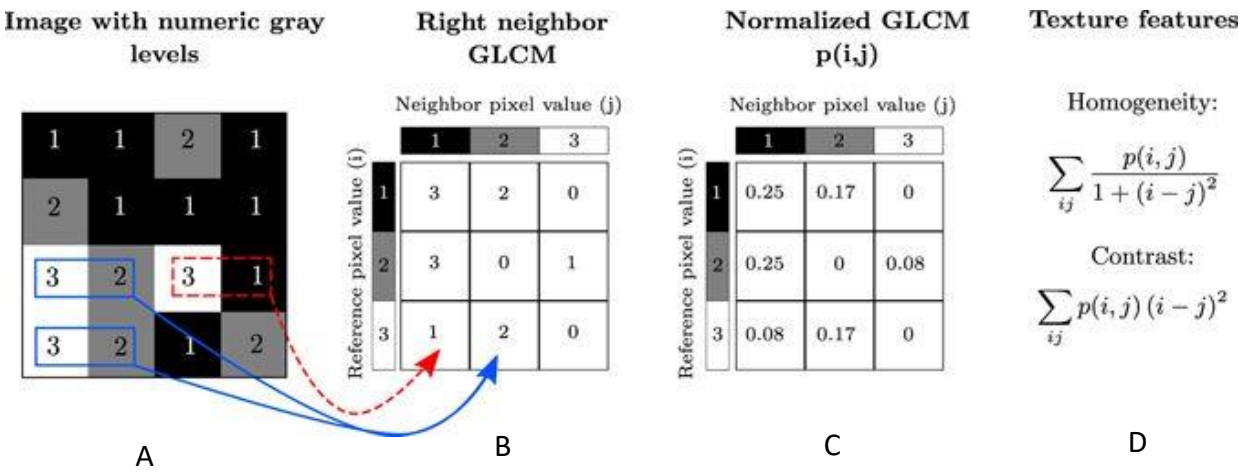


Figure 3.4: Starting from left to right, the first matrix (A) is an example of an input image consisting of 3 gray values. The second matrix (B) is the gray-level co-occurance matrix displaying the neighboring pixel combination frequencies. The third matrix (C) is the normalization of the grey-level co-occurance matrix. The equations (D) at the end are two of the thirteen Haralick properties that are computed from the normalized co-occurance matrix (C).

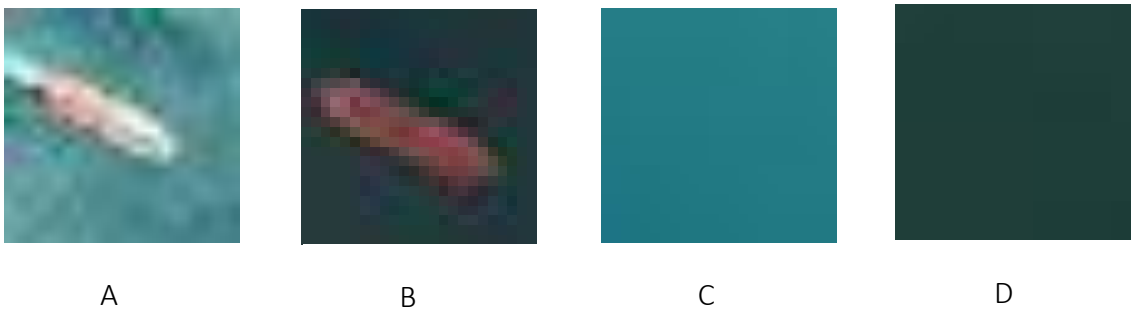


Figure 3.5: Starting from the left shows two clear positive samples (A and B) followed by two clear negative samples (C and D).

Figure 3.5 shows two distinctly positive samples and two distinctly negative samples from the dataset.

Table **3.2** displays the 13 Haralick properties of the 4 images in Figure 3.5. Each image generates 13 Haralick features using a python library called Mahotas [30].

Table 3.2: Starting from left to right, the first column represents the thirteen different Haralick properties. The other 4 columns are the 13 Haralick properties computed for the respective images from figure 3.4. The blue rows highlight 3 Haralick features showing large differences between the positive images (figure 3.5A and figure 3.5B) against the negative images (figure 3.5C and figure 3.5D).

Features	Figure 3.5A (+ve)	Figure 3.5B (+ve)	Figure 3.5C (-ve)	Figure 3.5D (-ve)
Angular Second Moment	0.001442	0.027341	0.074174	0.426959
Contrast	**536.726882**	**87.450201**	**0.312756**	**0.142348**
Correlation	0.679811	0.796231	0.968199	0.860976
Variance	**836.713522**	**213.804895**	**4.950022**	**0.516260**
Inverse Difference Moment	0.093960	0.442262	0.852058	0.937758
Sum Average	270.336248	108.765497	194.885732	104.310629
Sum Variance	**2810.127204**	**767.769379**	**19.487333**	**1.922693**
Sum Entropy	6.872061	5.398296	3.927014	1.856617
Entropy	9.610721	7.264246	4.254310	2.031945
Difference Variance	0.000146	0.001253	0.005757	0.014024

Difference Entropy	5.064218	3.589447	0.880117	0.560308
Information Measures of Correlation 1	-0.440144	-0.451964	-0.642322	-0.587387
Information Measures of Correlation 2	0.997791	0.992648	0.989792	0.898296

In

Table 3.2 there are 3 Haralick properties highlighted: contrast, variance, and sum variance. These 3 properties, as seen in the table, show a large range of differences from the positive samples and negative samples seen in Figure 3.5. These 3 properties show promising distinctions for ship detection while being computationally efficient.

3.2.3 Grayscale Histograms

Grayscale intensity histograms were used in our research to capture the distribution of grayscale pixel intensities. Histograms are a common feature in image processing and object detection. The idea behind evaluating the distribution of pixel intensities is that similar images will have similar distributions. In our research, we generated normalized histograms with 32 bins, 64 bins, 128 bins, and 256 bins using the calcHist function from the python library OpenCV [28]. The bins capture pixel intensity ranges, or in the case of the 256 bin histogram, can provide a one to one mapping of grayscale pixel intensity frequencies. Figure 3.6 shows a comparison of the different bin size histograms for a single positive sample. The 32 bin histogram in both the positive and

negative case captures the outline of the distribution while condensing the number of features or intensity values used. This reduces the computational complexity in the classifier training.

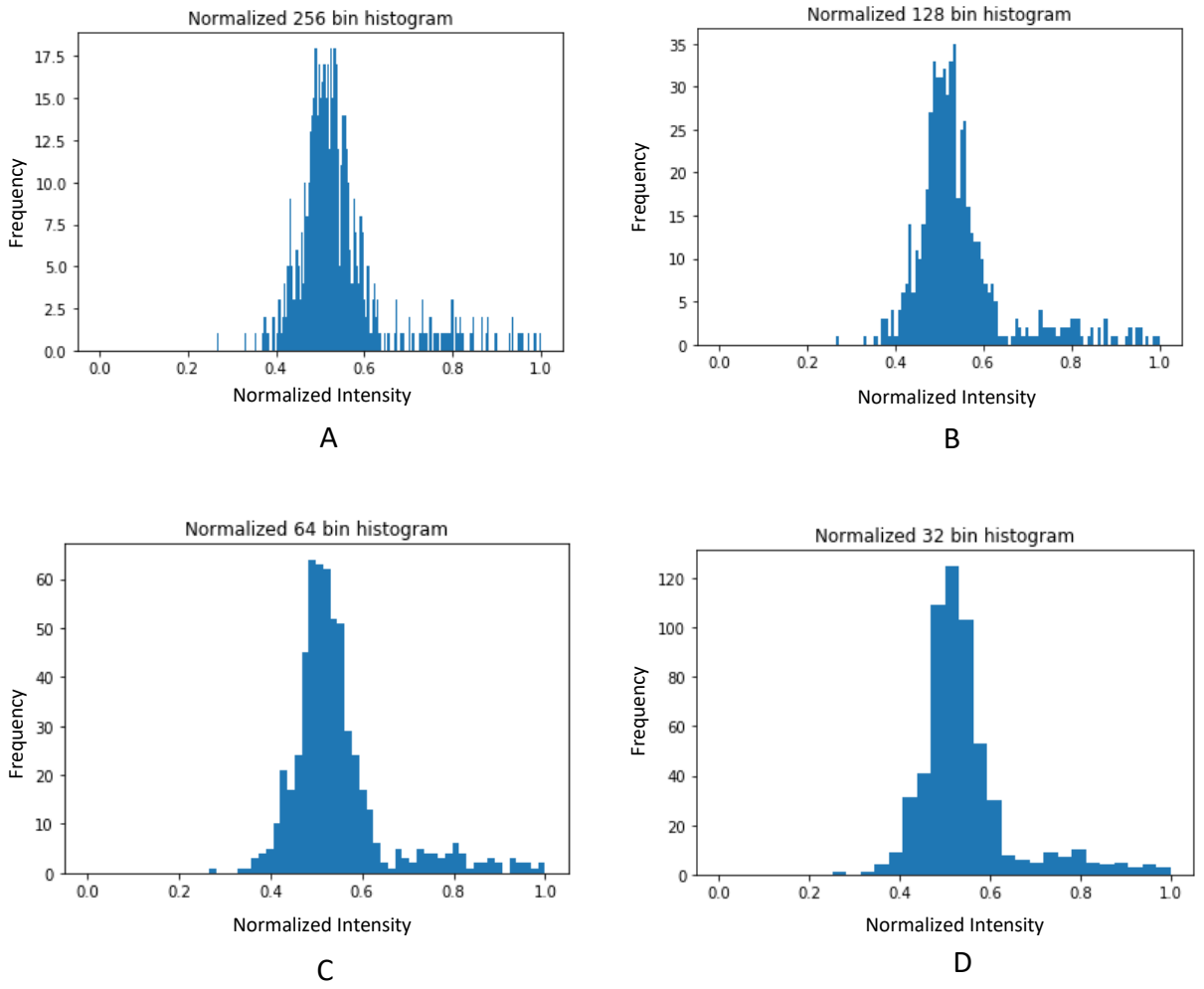


Figure 3.6: The 4 histograms displayed are the histograms of the positive sample in Figure 3.5A. Top left (A) shows the distribution of the positive sample using 256 bins. Top right (B) shows the pixel intensity distribution of the positive sample using 128 bins. Bottom left (C) shows the pixel intensity distribution of the positive sample using 64 bins. Bottom right (D) shows the pixel intensity distribution of the positive sample using 32 bins.

The positive sample histograms have a wider distribution than the negative sample histograms and have a tail to the right, suggesting a higher contrast in ship samples.

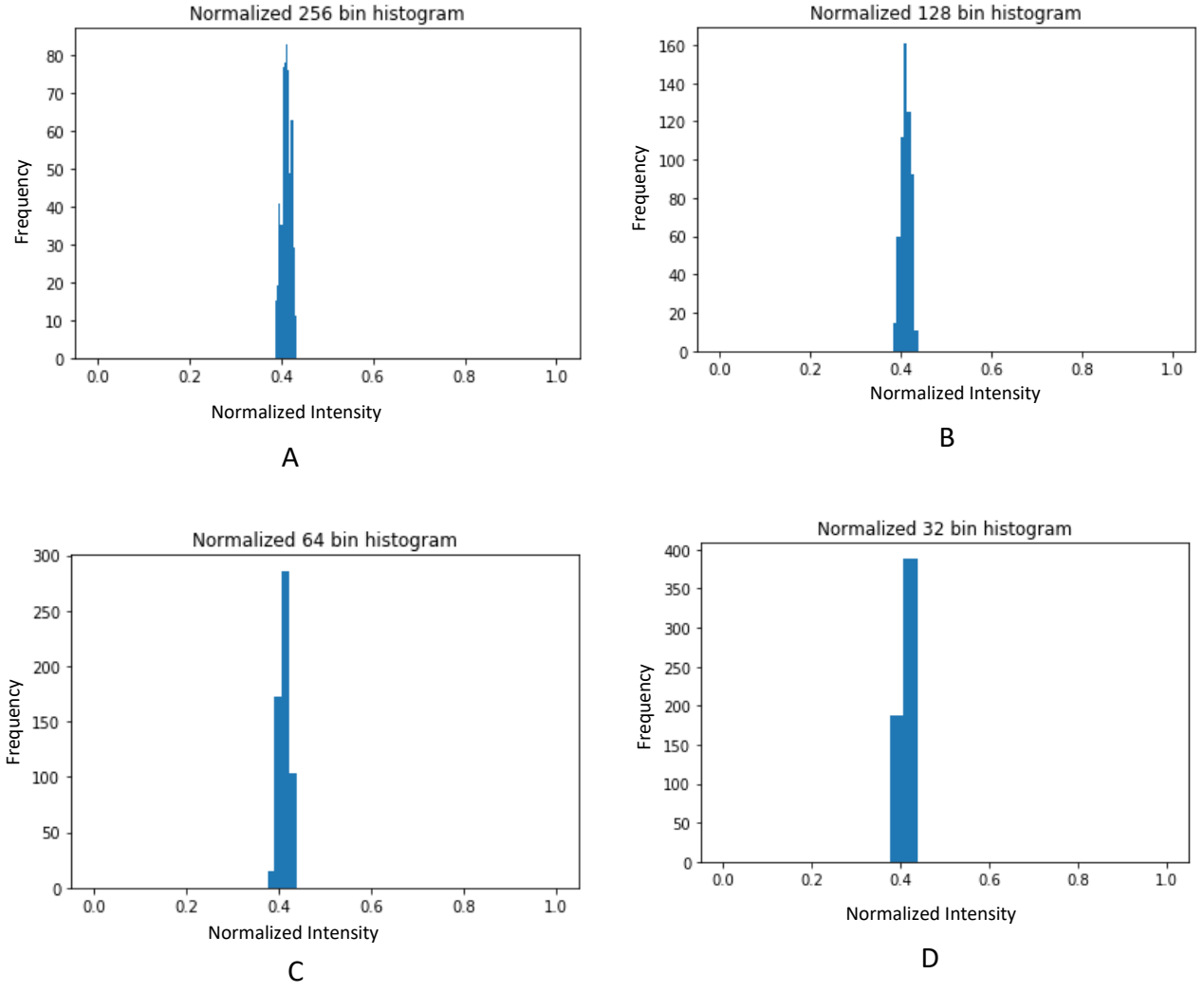


Figure 3.7: The 4 histograms displayed are the histograms of the negative sample in Figure 3.5C. Top left (A) shows the distribution of the negative sample using 256 bins. Top right (B) shows the pixel intensity distribution of the negative sample using 128 bins. Bottom left (C) shows the pixel intensity distribution of the negative sample using 64 bins. Bottom right (D) shows the pixel intensity distribution of the negative sample using 32 bins.

3.2.4 Histogram of Oriented Gradients (HOG)

The HOG feature is a popular descriptor used in computer vision and image processing. The technique works by counting the occurrences of gradient orientations within a localized portion of an image [31]. The image is divided into small regions or cells, in which a histogram of gradient directions is compiled for the pixels within a cell. Therefore shapes are described by the intensity of gradients or edge directions. Similar to edge detection, HOG features characterize local object appearances as well as shape in an image. Our dataset compared 16×16 pixel cell, 12×12 pixel cell, and 8×8 pixel cell sizes for the HOG descriptor. All ships in our imagery were less than 20 pixels in length. Using the different cell sizes, 16, 12 and 8, on our dataset produced 8, 32, and 72 HOG features, respectively. The HOG features were computed by the python library Scikit-Image, using their hog feature function [32]. Interestingly enough, the 16×16 window size with the least descriptors performed the best with our regression analysis averaging 88% accuracy alone. Figure 3.8 visualizes the different HOG descriptors based on window size.

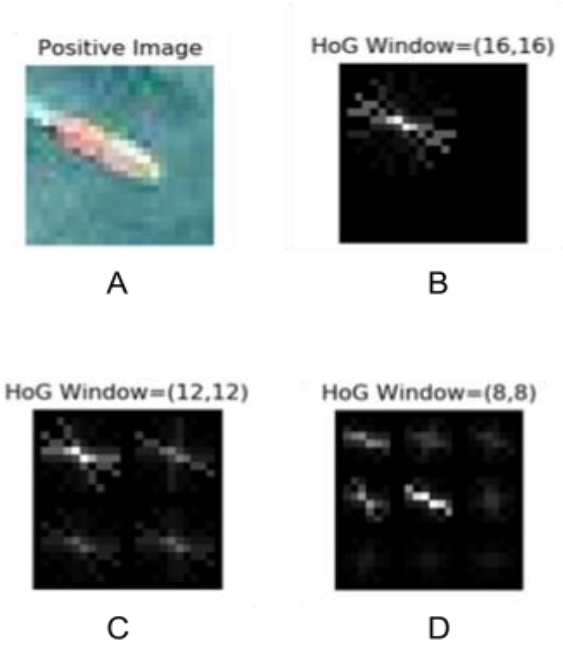


Figure 3.8: Top left image (A) is a positive sample from the dataset. Top right (B) is a visualization of the HOG feature for that positive sample using a window size of 16×16. Bottom left (C) is the visualization of the HOG feature using a 12×12 window. Bottom right (D) is the visualization of the HOG feature using a 8×8 window.

The HOG feature with a 16×16 window captures the gradient changes in the ship sample the best, as seen in figure 3.8B. As the window sizes decrease in Figure 3.8C and figure 3.8D, the local background gradient changes are picked up.

3.2.5 Local Binary Patterns

In 1996, T. Ojala *et al.* first described the texture operator known as Local Binary Patterns (LBPs) [33]. LBPs have since been used in computer vision as a powerful texture feature. LBPs have been used for face recognition [17], and C. Zhu *et al.* introduced LBPs or an extension of LBPs for vessel classification [18]. Moreover, X. Wang *et al.* propose that HOG and LBP descriptors together can increase accuracy for some datasets in classification problems [34]. LBP computations work by sub-sectioning an image into smaller windows and comparing a window's center pixel value with the values of its neighboring pixels. The comparison produces binary numbers for a window, which is then converted into decimal values. A histogram is then calculated overall window decimal values for an image. An extension to the LBP operator is the uniform pattern, which reduces the length of the feature vector and provides a rotation-invariant descriptor. In our research, we focus on the uniform LBP implementation, which creates 10 features for each image in our dataset using Scikit-Image [32]. Figure 3.9 shows an example of the LBP computation.

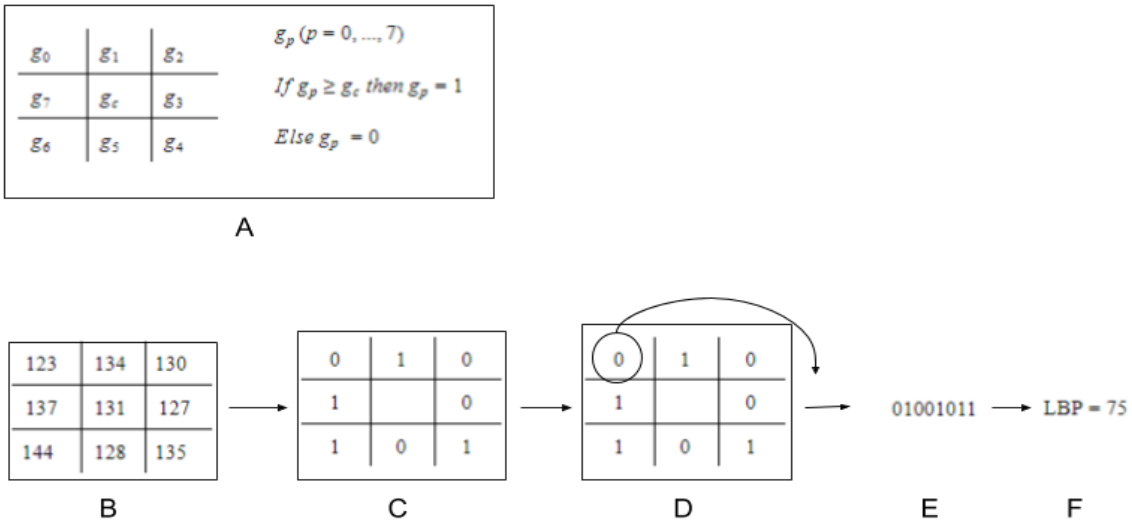


Figure 3.9: An example LBP calculation using a 3×3 window. (A) shows the notation for the window calculation where $g_p(p = 0, \dots, 7)$ represent the neighboring pixel values in the window and g_c represents the center pixel value. Matrix (B) is an example 3×3 window and matrix (C) is the binary matrix version of (B) after the neighboring pixel comparisons. Matrix (D) displays how the binary number is formed for the 3×3 and (E) is the resulting binary number. Lastly, (F) shows the decimal version of binary number (E).

In Figure 3.9 the neighboring pixel values (123, 134, 130, 127, 135, 128, 144, 137) are compared to the center pixel value (131). If the neighboring pixel values are greater than or equal to the center value, then the neighboring pixel value is changed to 1. Otherwise, the neighboring pixel value is changed to 0. Once all the neighboring pixel values are converted into 1 or 0, the neighboring pixel values are concatenated into a single binary number. This single binary number is computed by starting from the top-left neighboring pixel value and concatenating the next neighboring pixel value in a clockwise fashion.

3.2.6 Haar-like Features

Haar-like features were introduced by Paul Viola and Michael Jones for use in a real-time face detector [35]. Haar-like features are based on Haar wavelets and provide subsection categories for an image based on average pixel intensity differences among the different regions in an image. The advantage of Haar-like features over other features is computational efficiency during feature extraction. In our research we studied 5 Haar-like features: type-2-x, type-2-y, type-3-x, type-3-y, and type-4. The type 2-x and 2-y represent two rectangles varying in the x and y directions, respectively. Type 3-x and 3-y represent three rectangles varying in the x and y directions, respectively. Type 4 represents four rectangles varying along both the x and y-axis. A

window size of 12×12 pixels was used, which produced 10,228 features. Figure 3.10 shows the different Haar-like feature descriptors used from Scikit-Image [32].

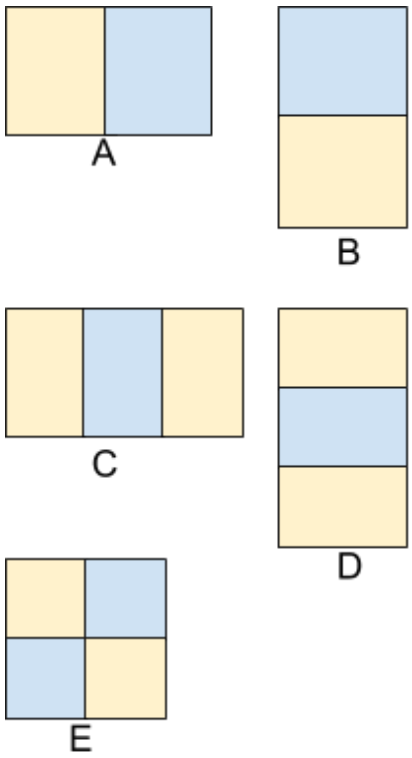


Figure 3.10: This figure shows the 5 different Haar-like window types used in this research. Figure 3.10A represents the Type-2-x Haar-like feature and contains two regions, this window calculates the horizontal change in the input image. Figure 3.10B represents the Type-2-y Haar-like feature and contains 2 regions, this window calculates the vertical change throughout the input image. Figure 3.10C represents the Type-3-x Haar-like feature and contains 3 regions, this window calculates the horizontal change throughout the input image. Figure 3.10D represents the Type-3-y Haar-like feature and contains 3 regions, this window calculates the vertical change throughout the input image. Lastly, figure 3.10E represents the Type-4 Haar-like feature and contains 4 regions, this window accounts for both the horizontal and vertical change throughout the input image.

Each window is placed in all possible locations of the input image. At each location, the yellow and blue region's pixel intensity values are summed up independently. The difference of the yellow region's sum from the blue region's sum is calculated and produces a single-valued feature

for that window at a specific location. The window acts like a sliding window throughout the input image in order to calculate the difference feature at all locations. Figure 3.10A and Figure 3.10B represent the two regions sliding window Haar-like feature. These two features are commonly referred to as edge features. The difference of an edge region will be much larger than the difference between a background or a flat region. Figure 3.10C and Figure 3.10D represent a 3 region window feature and are commonly referred to as line features. Figure 3.10E is the 4 regions rectangle Haar-like feature.

3.3 Feature Extraction

There are 15 total individual features and variations used in this research: Hu moments, Haralick features, HOG 8×8, HOG 12×12, HOG 16×16, Histogram 32-bin, Histogram 64-bin, Histogram 128-bin, Histogram 256-bin, LBP features, Haar-like type-2-x, Haar-like type-2-y, Haar-like type-3-x, Haar-like type-3-y, and Haar-like type-4. A feature text file is created for each feature over the entire dataset of 2955 images. Each feature file consists of 2955 rows where each row represents an image in our dataset. The rows contain the binary classification value and the comma-separated feature values depending on the feature extracted. In the case of the LBP feature type in which 10 feature values are generated per image, the feature file would contain 2955 rows with 11 commas separated values per row. The first value of each row is the classification value, i.e., 0 for not a ship and 1 for a ship. In the case of combined features, each feature's value is appended to the end of the row after the previous feature extraction. For example, the combined features Hu moments and LBP would create a feature file of 2955 rows

where each row contained 18 commas separated values. Again, the first value being the classification value. This research incorporates an additional 9 combined features. Thus the total number of feature variations and combinations used was 24.

Various image processing and computer vision libraries were used for feature extraction, as mentioned previously. Every image was converted into grayscale before each feature extraction, and in the case of Haar-like feature extraction, the images were further converted into integral images for computational efficiency. For use in the machine learning algorithms, each feature file was read into a NumPy array, and the first column was extracted for the true Y values.

3.4 Machine Learning Classifiers

The following classifiers from Scikit-learn [36] were used for our feature analysis: Support Vector Machine, Logistic Regression, Random Decision Forest, Extra Trees Classifier, K Nearest Neighbors, Bagging, Gradient Boosting Classifier and eXtreme Gradient Boosting Classifier. Each algorithm was implemented with 10-fold cross-validation. 10-fold cross-validation splits the dataset into 10 groups, where 9 groups are used for training and 1 group for validation. This method is repeated 10 times so that each group will be the validation set. The performance results from each repetition are then averaged to provide the overall classification results. A further discussion of each algorithm follows.

3.4.1 Support Vector Machine (SVM)

Cortes and Vapnik introduced the standard SVM in 1995 as a supervised machine learning technique for a two-group classification problem [37]. The idea behind SVM is to find an optimal hyperplane after the input vector is non-linearly mapped to a high-dimensional feature space. SVMs can be used for classification or regression problems and remain computationally efficient. SVM has been widely used in the literature for either preprocessing [11], in conjunction with other algorithms such as neural networks [38], and as a main classifier [39].

3.4.2 Logistic Regression (LG)

Joseph Berkson is said to be the developer of logistic regression as a general statistical model in 1944, but the logistics function dates back to the 1800s [40]. LG is a linear model that attempts to model a binary dependent variable through the use of a sigmoid function. It provides a probability for classification. Corbane *et al.* implement a logistic model for their ship detection research using wavelets and radon transform [10]. Moreover, Tang *et al.* add a logistic regression layer after their deep neural network as a fine-tuning step [41].

3.4.3 Random Decision Forest (RDF)

RDFs are an ensemble learning technique that constructs a multitude of decision trees for classification and regression. For classification, the decision trees output a class, and the mode

of the classes is used to determine the final class. The algorithm was first introduced by Tin Kam Ho in 1995 [42] and later extended to include bagging by Leo Breiman in 2001 [43]. Huang *et al.* and Dong *et al.* both create an extension of the random forest to perform target detection [38], [39].

3.4.4 Extra Trees Classifier (ETC)

ETCs were introduced by Geurts *et al.* in 2006 [44]. The extremely randomized trees or ExtraTrees classifier is an adaptation of random forest with a few noticeable differences. ETCs are an ensemble of individual trees similar to RDFs, but in ETCs, each tree is trained on the whole learning sample, and the top-down splitting in the tree learner is randomized.

3.4.5 K-Nearest Neighbors (KNN)

KNN is a nonparametric pattern recognition algorithm for classification and regression [45]. In most cases, Euclidean distance is used as a measure between objects in order to determine the k-nearest neighbors. In classification, a plurality vote is used among the k-neighbors of an object to determine the class. In the literature, Huang *et al.* used KNN as a comparison to their modified RDF algorithm [46], and Gallego *et al.* combined a Convolutional Neural Network (CNN) with the KNN algorithm for improved performance [13].

3.4.6 Gradient Boosting Classifier (GBC)

Breiman introduced the idea of gradient boosting in terms of an optimization algorithm for a cost function [47]. Gradient boosting is a machine learning technique that produces an ensemble of weak prediction models, such as decision trees. The models are additive in that results from previous models will change the gradients of the next model in order to optimize classification. The idea is that multiple weak classifiers can iteratively build a strong classifier.

3.4.7 Extreme Gradient Boosting Classifier (XGBC)

EXtreme Gradient Boosting (XGB) started as a research project by Chen in 2016 to create a scalable gradient boosting algorithm [48]. XGB builds from GB but incorporates advanced regularization as well as computes second-order gradients of the loss function. The advanced regularization, L1 and L2, improves model generalization and further reduces overfitting. Moreover, the second-order gradients provide more information on gradient direction and improve loss function minimization.

3.4.8 Bagging (BAG)

Bagging, formerly known as bootstrap aggregating, is an ensemble meta-algorithm designed for improved stability in classification as well as regression. Bagging was originally developed by Leo Breiman in 1994 [49] and later incorporated in his Gradient Boosting technique. Bagging usually

implements a decision tree method but can use any method such as SVM as its base estimator. The idea is for the base estimator to fit on random subsets of the dataset and then aggregate the predictions to form a final decision. This randomization ensemble typically reduces variance with black-box estimators such as decision trees.

4. Results

Results for our dataset consisting of 955 positive samples and 2000 negative samples are now discussed. After individual feature type evaluation, the top-performing feature types were chosen for further analysis in different combinations.

4.1 Feature Comparison

We evaluated 6 features with parameter variations in 3 features, totaling 15 features. All 15 features were trained individually using the 8 classification algorithms. *Precision*, *recall*, and *F1 score* are the main evaluation metrics employed in this study. Precision, recall, and F1 score are defined in Table 4.1.

Table 4.1: Name and definition of performance evaluation metrics.

Name of Metric	**Definition**
True Positive (TP)	Correctly predicted ship images
True Negative (TN)	Correctly predicted non-ship images
False Positive (FP)	Incorrectly predicted ship images
False Negative (FN)	Incorrectly predicted non-ship images

Recall/True Positive Rate (TPR)	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$
Specificity /True Negative Rate (TNR)	$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$
Accuracy (ACC)	$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{FP}+\mathrm{TP}+\mathrm{TN}+\mathrm{FN}}$
Precision	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$
F1 score	$\frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$

Precision and recall are better indicators of performance than accuracy when the dataset is asymmetric because they both take into account the type of prediction. Precision calculates the proportion of predicted ships over the number of actual ships. Recall calculates the proportion of actual ships that are correctly classified as ships. Accuracy, on the contrary, calculates the total number of correctly predicted ships and non-ships overall ship/non-ship samples. Hence, if the classes are unevenly distributed, then one class can significantly skew the performance measure. This study also uses an F1 score because it calculates the weighted average of precision and recall and thus provides a balance between the two metrics. These metrics were chosen due to having

an asymmetric dataset but also because they are commonly used in similar analyses [3]–[6], [21]. The 14 features and each classifier's precision score is plotted in Figure 4.1.

The best individual feature type was the Haar-like features, which focus on local, regional intensity differences. The Haar-like feature type had high precision scores between 90%-96% for

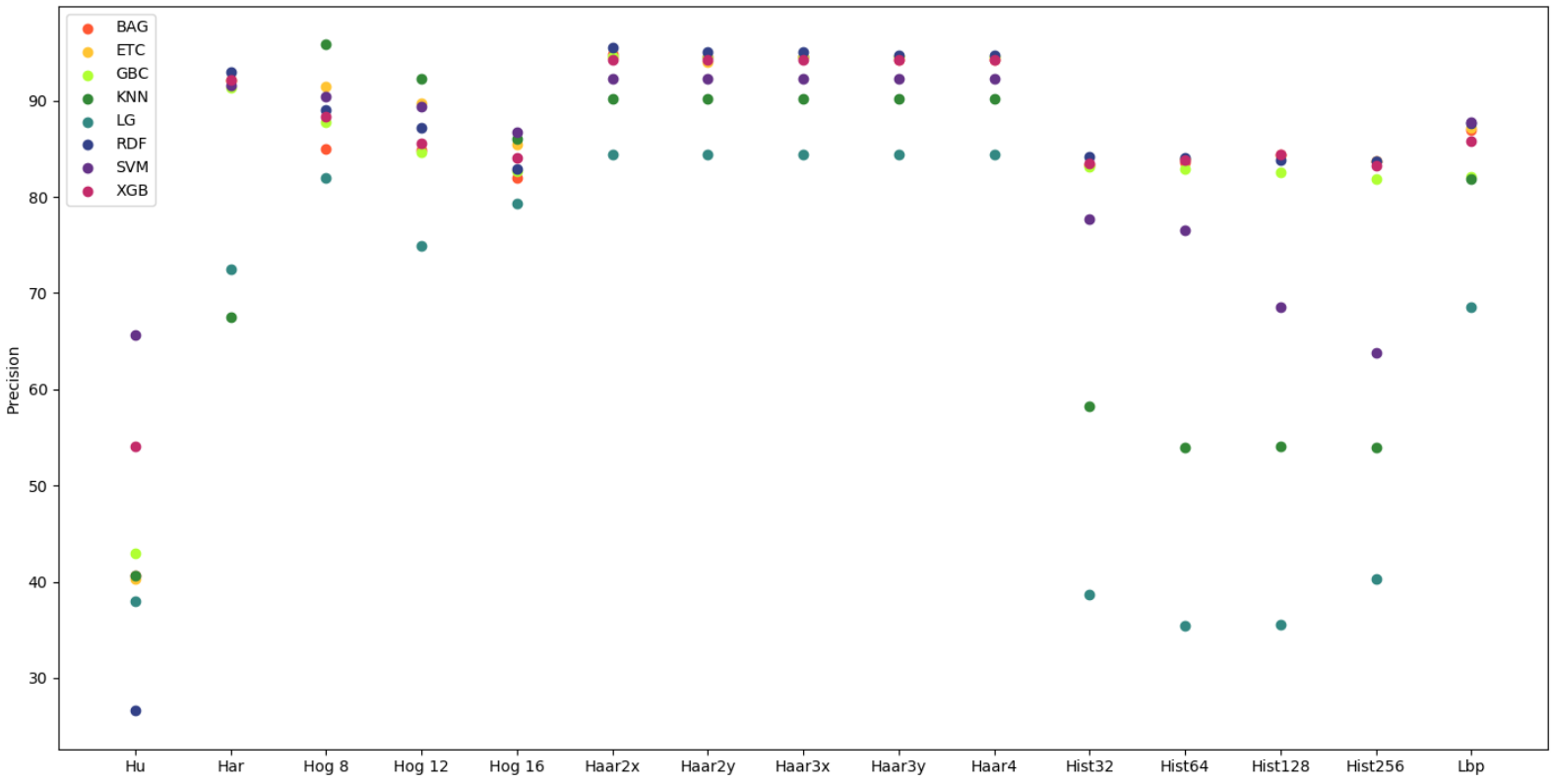


Figure 4.1: The x-axis represents the 14 individual features and the y-axis represents the precision score. For each feature the precision scores for each 8 classification algorithm is plotted.

most of the classifiers, the best precision being 95.56% for Haar-like type-2-x using RDF classification. LG classification had the lowest precision scores for the Haar-like features, where scores fell in the 83%-85% range. Although the Haar-like feature types performed well, the 10,228 feature vector size incurred a high computational cost. We continued further feature

analysis with the Haar-like type-2-x feature in our feature combination analysis. The next best feature type was the Haralick features, with most classifiers reaching between 91%-93% precision. Haralick features achieved a max precision of 92.93% through RDF classification. Among the low performing classifiers for the Haralick feature type were LG and KNN, reaching precision scores of 72.43% and 67.52%, respectively. The LBP feature type had an average precision of 83.64% among the 8 classifiers, with SVM performing the best at 87.80% precision and LG performing the worst at 79.33%.

Figure 4.1 shows the HOG feature type with a window size of 8x8 performs the best out of the HOG features, but this feature is misleading when looking at precision alone. An analysis of the precision-recall curve for each HOG variation shows better performance with the larger window HOG feature, HOG 16. Therefore, we used HOG 16 for further feature analysis. Figure 4.2 shows the three HOG precision-recall curves. HOG 16 had an average precision score of 83.64%, with SVM reaching the highest precision of 86.71% and LG scoring a low 79.33% precision.

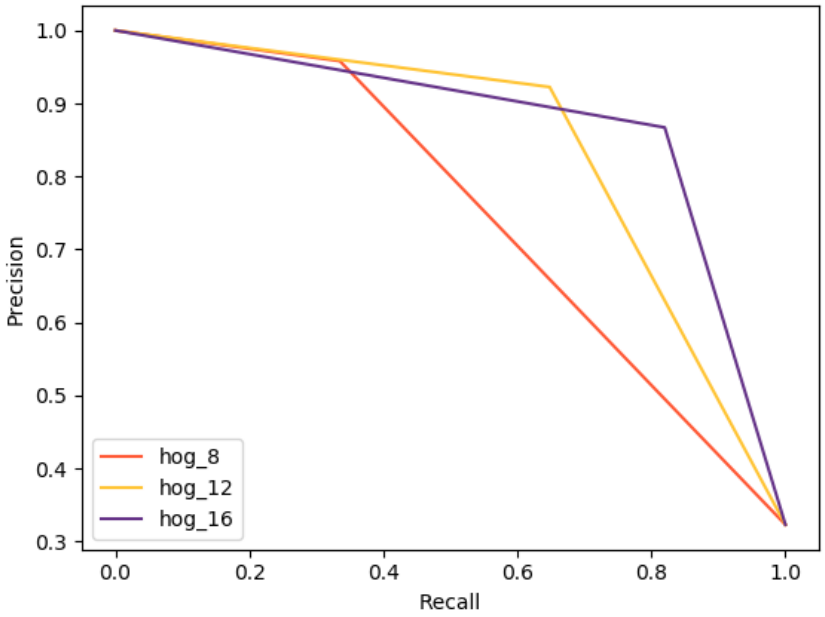


Figure 4.2: The graph shows the precision and recall balance between the 3 variations of HOG features. The hog_16 feature provides the best balance between precision and recall.

Among the histogram feature type, the 256-bin histogram has better precision performance, but the 32-bin histogram feature has slightly better precision-recall performance as the precision-recall curves in Figure 4.3 indicate. Histogram 32-bin averaged 74.01% precision, with RDF scoring the highest precision of 84.18% and LG scoring the lowest precision at 38.67%.

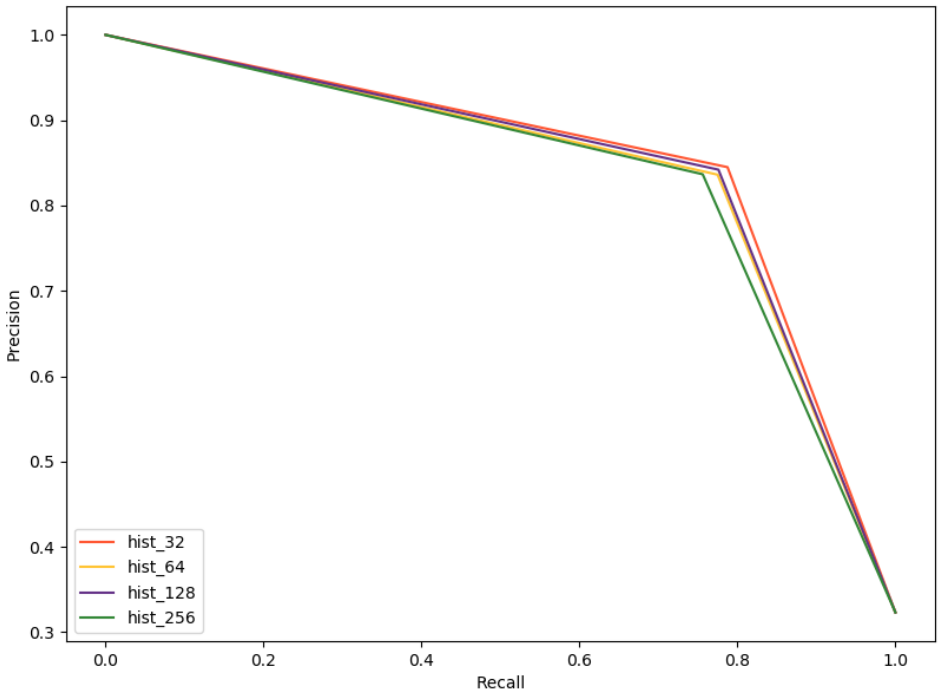


Figure 4.3: The graph shows the precision and recall balance between the 4 variations of grayscale histogram features. The hist_32 feature provides the best balance between precision and recall.

Lastly, the least efficient predictor was the Hu moments with a 43.62% average precision, significantly below the other features. This could be due to the low resolution of our images, which can cause Hu moment performance to decline [27]. Hu moments were consequently removed from further feature analysis.

Although the F1 score is our primary metric, we also measure specificity, accuracy, precision, and recall for each feature. Of note, the Haralick features had the best average F1 score of all 14 features, while Haar-like Type-2-X had the best average precision and second-best averaged F1 score. The Haar-like type-2-x had the best average F1 score among the Haar-like feature types. The HOG 16 feature had the best average F1 score among the other HOG variations. The 32-bin histogram feature had the best average F1 score among the histogram variations. Lastly, the Hu moments had the lowest performance among all 14 features, with an averaged F1 score of 25.71%. Table 4.2 shows a summary of the 8 classifiers averaged precision, recall, and F1 scores for each feature type and variation. We also note the number of features, as that can hinder computational efficiency.

Table 4.2: Averaged metrics across all 8 classifiers for each of the 14 individual features. The haralick feature had the highest average F1 score.

Feature	Number of Features	Specificity	Accuracy	Precision	Recall	F1 Score
Hu Moments	7	85.94%	65.50%	43.62%	22.70%	25.71%
Haralick	13	93.64%	90.41%	86.49%	83.64%	**84.99%**
LBP	10	93.42%	86.26%	83.48%	71.27%	**76.80%**
Histogram 32-bin	32	90.29%	82.56%	74.01%	66.35%	**69.35%**
Histogram 64-bin	64	90.08%	81.60%	73.02%	63.84%	67.50%
Histogram 128-bin	128	89.56%	81.24%	72.20%	63.81%	67.22%
Histogram 256-bin	256	89.31%	80.56%	71.78%	62.24%	66.23%

HOG 8	72	95.45%	87.50%	88.70%	70.86%	77.47%
HOG 12	32	94.26%	87.80%	86.07%	74.27%	79.57%
HOG 16	8	92.79%	87.94%	83.64%	77.80%	**80.44%**
Haar 2-x	10228	97.16%	90.55%	**92.55%**	76.71%	**83.45%**
Haar 2-y	10228	97.09%	90.50%	92.40%	76.69%	83.37%
Haar 3-x	10228	97.13%	90.51%	92.47%	76.66%	83.39%
Haar 3-y	10228	97.09%	90.52%	92.40%	76.74%	83.40%
Haar 4xy	10228	97.08%	90.49%	92.37%	76.69%	83.36%

Lastly, we briefly comment on the classification algorithms and their predictive performance. Figure 4.4 plots the classifier performance according to precision for the top 6 feature types.

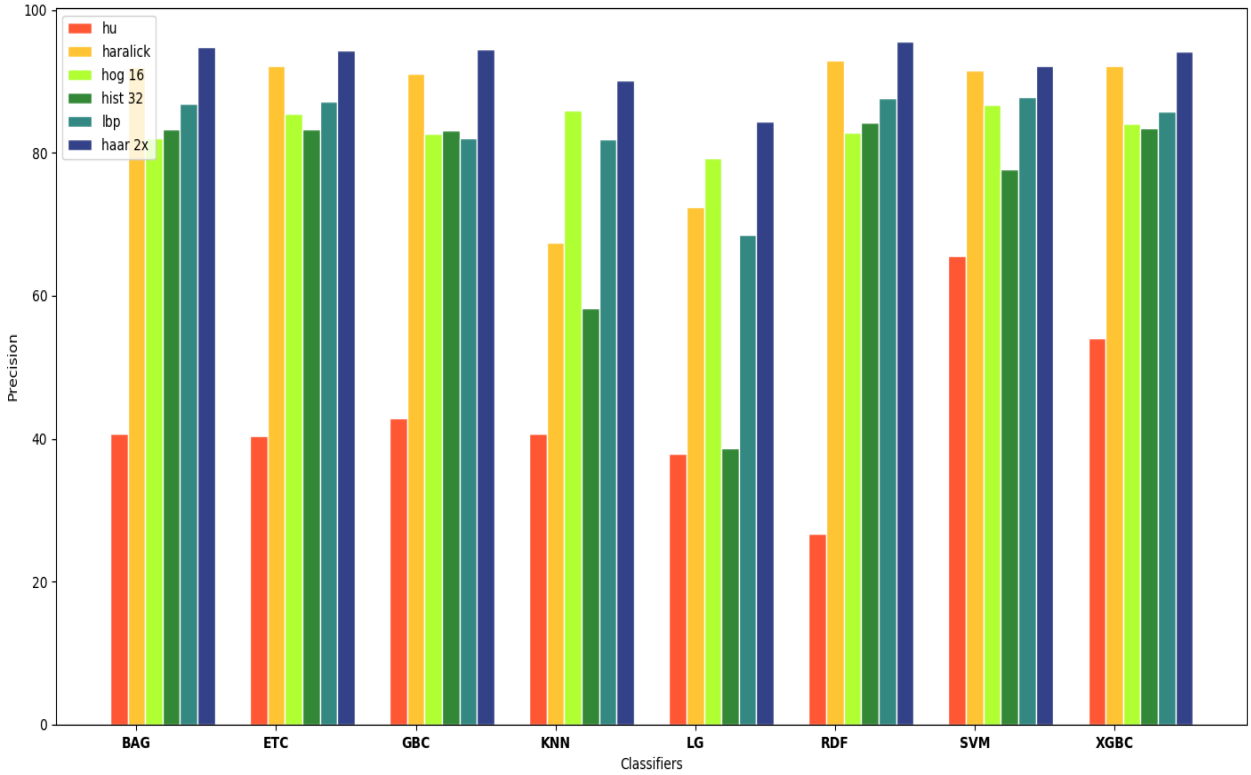


Figure 4.4: Precision scores for Hu moments, Haralick, HOG 16, Histogram 32-bin, LBP and Haar-like type-2-x features by the individual classifiers. This plot demonstrates classifier performance for the 6 feature types.

As previously mentioned, the Haar-like feature type and Haralick feature type perform the best. The precision scores are fairly consistent across BAG, ETC, GBC, RDF, SVM, and XGBC. In almost all cases, the poorest classifiers were KNN and LG. If we disregard KNN and LG, our performance improves considerably.

4.2 Combined Features

After individual feature comparisons, a few feature type combinations were tested with the 8 classifiers. Our base feature became the Haralick feature over the Haar-like feature because of the better precision/recall performance and computational efficiency. In terms of real-time or human-in-the-loop processing time complexity becomes important. The haar-like features in some cases required hours of training time. In contrast, the Haralick features worst case training time was 150 seconds. Table 4.3 shows classifier training times in seconds for the Haralick and Haar-like type-2-x features.

Table 4.3: Training times by classifier for the Haar-like Type-2-x feature and the Haralick feature. The Haar-like features, in some cases, are computationally infeasible for real-time processing.

Feature	BAG	ETC	GBC	KNN	LG	RDF	SVM	XGB	Average
Haar 2-x	49145 (sec)	588 (sec)	14121 (sec)	71 (sec)	579 (sec)	619 (sec)	4846 (sec)	679 (sec)	8831 (sec)
Haralick	**150 (sec)**	31 (sec)	91 (sec)	5 (sec)	5 (sec)	59 (sec)	37 (sec)	13 (sec)	49 (sec)

We excluded Hu moments due to their poor performance individually but included LBP as Wang *et al.* suggest performance improvements when paired with HOG features [34]. We note that the combinations are not a robust analysis of all possible feature combinations but were selected based on individual performance. The 9 feature combinations studied were:

- Haralick, Haar Type-2-X
- Haralick HOG 16
- Haralick, LBP
- HOG 16, LBP
- Haralick, HOG 16, LBP
- Haralick, HOG 16, Histogram 32
- Haralick, HOG 16, Haar Type-2-X
- Haralick, HOG 16, Haar Type-2-X, LBP
- Haralick, HOG 16, Histogram 32, LBP

Figure 4.5 visualizes the precision scores for each combination by the classifier. The best average precision of 93.72% can be seen from the Haralick with Haar-like features combination. Haralick and Haar-like features combination had the best precision score of 96.95% using GBC. Following Haralick and Haar-like features, the next best average precision was the Haralick, HOG 16, and Haar-like features combination at 93.05%. This combination also had the best precision of 96.95% using GBC. Average results for all feature combinations can be summarized in Table 4.4. The least performing combination with an average precision of 88.33% was Haralick and LBP features. Even though this combination performed poorly compared to the other feature combinations, the best precision score for this feature combination still reached 93.64% with RDF classification. All feature combinations had multiple classifiers reaching scores in the ’90s. Lastly, as was seen with the individual feature analysis, the lessor performing classifiers were KNN and LG for all feature combinations.

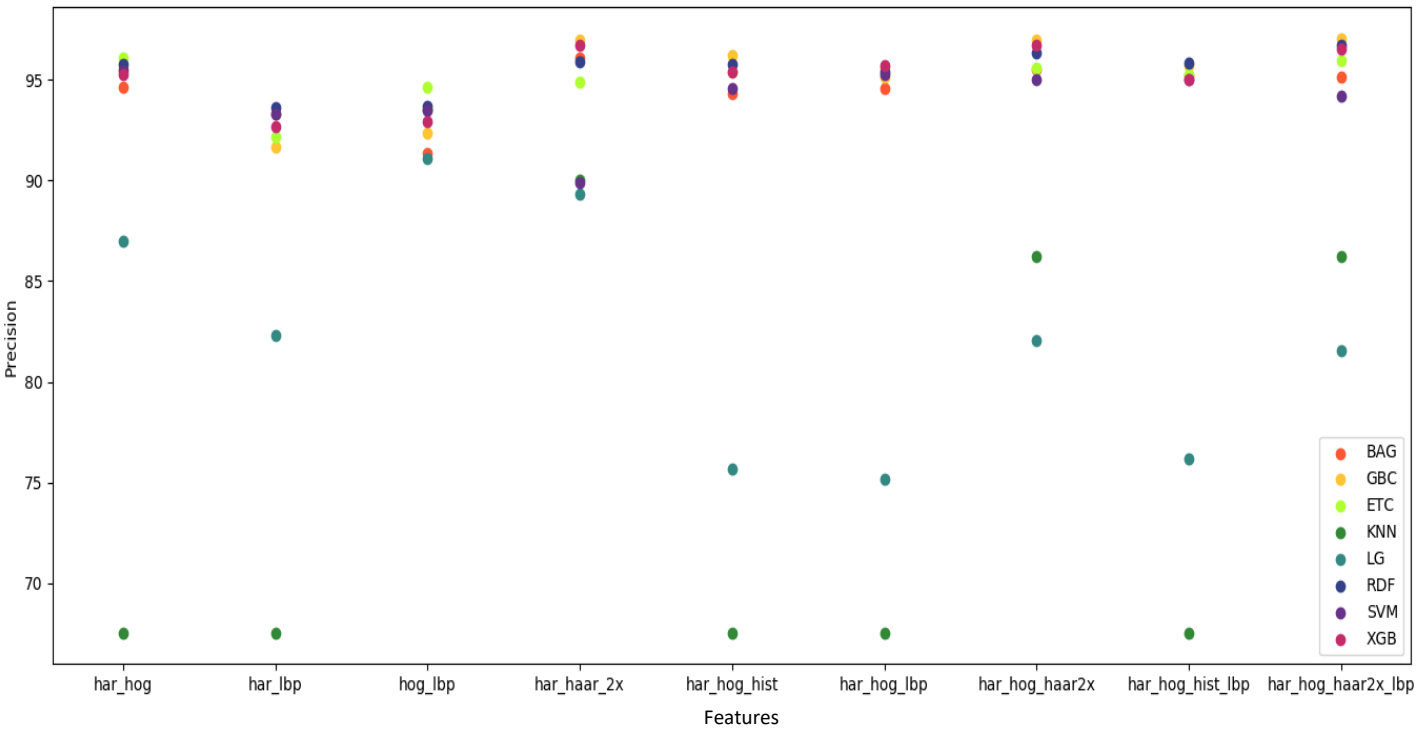


Figure 4.5: The x-axis represents the 9 different combination features and the y-axis represents the precision score. For each feature the precision scores for each 8 classification algorithm is plotted.

Table 4.4 below shows the average classifier accuracy for 9 different feature combinations. Although Haralick with Haar-like features had the best average precision, this combination did not have the best recall/precision ratio or average F1 score. The best average F1 score was with the HOG 16 and Haralick features, combination 2 below.

Table 4.4: Averaged metrics across all 8 classifiers for each of the 9 different combination features. The haralick and HOG 16 combination had the highest average F1 score, but the highest precision was the combination of Haralick and Haar-like Type-2-x features.

#	Feature	Number of Features	Specificity	Accuracy	Precision	Recall	F1 Score
1	Haralick, Haar 2-x	10241	97.59%	91.78%	**93.72%**	79.61%	85.61%
2	Haralick, HOG 16	21	95.63%	92.86%	90.91%	87.04%	**88.88%**
3	Haralick, LBP	23	94.45%	91.28%	88.33%	84.63%	86.39%
4	HOG 16, LBP	18	96.96%	92.61%	92.89%	83.51%	**87.83%**
5	Haralick, HOG 16, LBP	31	94.89%	91.86%	89.29%	85.51%	87.30%
6	Haralick, HOG 16, Histogram 32-bin	53	94.99%	91.62%	89.36%	84.58%	86.84%
7	Haralick, HOG 16, Haar 2-x	10249	97.18%	92.21%	93.05%	81.79%	86.92%
8	Haralick, HOG 16, Haar 2-x, LBP	10259	97.11%	92.32%	92.92%	82.30%	**87.15%**
9	Haralick, HOG 16, LBP, Histogram 32-bin	63	95.06%	91.59%	89.47%	84.35%	86.76%

Combinations 4, 5, and 8 reach average F1 scores just under combination 2, with combination 8 and 4 reaching better average precision scores of 92.92% and 92.89%, respectively.

5. Conclusion

Multiple features and combinations were studied using multiple classifiers for ship detection in optical satellite imagery. A curated dataset that consisted of 955 positive samples and 2000 negative samples, including multiple environments such as near shore, clouds, ship wakes, and islands, were created. There were 24 different features and combinations studied, and 8 different classification algorithms implemented. The focus of this research was on feature analysis for explainability and performance, specifically in limited datasets and human-in-the-loop applications. The feature combination of Haralick, Haar-like, HOG, and LBP features achieved the best metrics with 97.07% precision, 93.51% recall, and a 95.25% F1 score. The best feature combination though in terms of performance and computational efficiency was the Haralick, HOG, and Histogram combination, which achieved a precision of 96.18%, a recall score of 92.36%, and a 94.23% F1 score. Moreover, removing the Haar-like feature only slightly lowered performance metrics. The Haralick features, which have not been used in ship detection previously, were the best performers in terms of the balance between precision and recall.

In future studies, feature analysis of LMP features, as well as implementing the top-performing features in this study for human-in-the-loop training applications, would be pursued. Also, an in-depth analysis of Haralick properties for ship detection could provide better means for

explainability. Lastly, I'd like to provide a comparison of the features within this research against the state-of-the-art algorithms.

References

[1] U. Kanjir, H. Greidanus, and K. Oštir, “Vessel detection and classification from spaceborne optical images: A literature survey,” *Remote Sens. Environ.*, vol. 207, no. November 2016, pp. 1–26, 2018.

[2] S. Iwin Thanakumar Joseph, J. Sasikala, and D. Sujitha Juliet, “Ship detection and recognition for offshore and inshore applications: a survey,” *Int. J. Intell. Unmanned Syst.*, no. September, 2019.

[3] Guang Yang, Bo Li, Shufan Ji, Feng Gao, and Qizhi Xu, “Ship Detection From Optical Satellite Images Based on Sea Surface Analysis,” *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 3, pp. 641–645, 2013.

[4] S. Qi, J. Ma, J. Lin, Y. Li, and J. Tian, “Unsupervised Ship Detection Based on Saliency and S-HOG Descriptor From Optical Satellite Images,” *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 7, pp. 1451–1455, 2015.

[5] R. Zhang, J. Yao, K. Zhang, C. Feng, and J. Zhang, “S-Cnn-based ship detection from high-resolution remote sensing images,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. - ISPRS Arch.*, vol. 41, no. June, pp. 423–430, 2016.

[6] N. Wang, B. Li, Q. Xu, and Y. Wang, “Automatic ship detection in optical remote sensing images based on anomaly detection and SPP-PCANet,” *Remote Sens.*, vol. 11, no. 1, 2019.

[7] F. Wu, Z. Zhou, B. Wang, and J. Ma, “Inshore Ship Detection Based on Convolutional Neural Network in Optical Satellite Images,” *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 11, no. 11, pp. 4005–4015, 2018.

[8] H. Lin, Z. Shi, and Z. Zou, “Fully Convolutional Network with Task Partitioning for Inshore Ship Detection in Optical Remote Sensing Images,” *IEEE Geosci. Remote Sens. Lett.*, vol. 14, no. 10, pp. 1665–1669, 2017.

[9] G. Liu, Y. Zhang, X. Zheng, X. Sun, K. Fu, and H. Wang, “A New Method on Inshore Ship Detection in High-Resolution Satellite Images Using Shape and Context Information,” *IEEE Geosci. Remote Sens. Lett.*, vol. 11, no. 3, pp. 617–621, Mar. 2014.

[10] C. Corbane, L. Najman, E. Pecoul, L. Demagistri, and M. Petit, “A complete processing chain for ship detection using optical satellite imagery,” *Int. J. Remote Sens.*, vol. 31, no. 22, pp. 5837–5854, 2010.

[11] S. Zhang, R. Wu, K. Xu, J. Wang, and W. Sun, “R-CNN-Based Ship Detection from High Resolution Remote Sensing Imagery,” *Remote Sens.*, vol. 11, no. 6, p. 631, 2019.

[12] X. Li, S. Wang, B. Jiang, and X. Chan, “Inshore ship detection in remote sensing images based on deep features,” *2017 IEEE Int. Conf. Signal Process. Commun. Comput. ICSPCC 2017*, vol. 2017-Janua, pp. 1–5, 2017.

[13] A. J. Gallego, A. Pertusa, and P. Gil, “Automatic ship classification from optical aerial images with convolutional neural networks,” *Remote Sens.*, vol. 10, no. 4, pp. 1–20, 2018.

[14] Q. Shi, W. Li, R. Tao, X. Sun, and L. Gao, “Ship Classification Based on Multifeature Ensemble with Convolutional Neural Network,” *Remote Sens.*, vol. 11, no. 4, p. 419, 2019.

[15] V. Buhrmester, D. Münch, and M. Arens, “Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey,” no. Ml, pp. 1–22, 2019.

[16] M. Pietikäinen, “Image analysis with local binary patterns,” *Image Anal. Proc.*, vol. 3540, pp. 115–118, 2005.

[17] V. F. Arguedas, “Texture-based vessel classifier for electro-optical satellite imagery,” in *2015 IEEE International Conference on Image Processing (ICIP)*, 2015, pp. 3866–3870.

[18] C. Zhu, H. Zhou, R. Wang, and J. Guo, “A novel hierarchical method of ship detection from spaceborne optical image based on shape and texture features,” *IEEE Trans. Geosci. Remote Sens.*, vol. 48, no. 9, pp. 3446–3456, 2010.

[19] J. Antelo, G. Ambrosio, J. González, and C. Galindo, “Ship detection and recognition in high-resolution satellite images,” *Int. Geosci. Remote Sens. Symp.*, vol. 4, pp. 514–517, 2009.

[20] S. Qi, J. Ma, C. Tao, C. Yang, and J. Tian, “A Robust Directional Saliency-Based Method for Infrared Small-Target Detection Under Various Complex Backgrounds,” *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 3, pp. 495–499, May 2013.

[21] Z. Zou and Z. Shi, “Ship Detection in Spaceborne Optical Image with SVD Networks,” *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 10, pp. 5832–5845, 2016.

[22] T. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, “PCANet: A Simple Deep Learning Baseline for Image Classification?,” *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5017–5032, Dec. 2015.

[23] Y. Yu, H. Ai, X. He, S. Yu, X. Zhong, and M. Lu, “Ship Detection in Optical Satellite Images Using Haar-like Features and Periphery-Cropped Neural Networks,” *IEEE Access*, vol. 6, pp. 71122–71131, 2018.

[24] “DigitalGlobe.” [Online]. Available: https://www.digitalglobe.com/products/satellite-imagery. [Accessed: 07-Apr-2020].

[25] S. Qiu, “BBox-Label-Tool,” 2017. [Online]. Available: https://github.com/puzzledqs/BBox-Label-Tool. [Accessed: 25-Mar-2020].

[26] Ming-Kuei Hu, “Visual pattern recognition by moment invariants,” *IRE Trans. Inf. Theory*, vol. 8, no. 2, pp. 179–187, Feb. 1962.

[27] Z. Huang and J. Leng, “Analysis of Hu’s moment invariants on image scaling and rotation,” *ICCET 2010 - 2010 Int. Conf. Comput. Eng. Technol. Proc.*, vol. 7, no. May 2010, 2010.

[28] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s J. Softw. Tools*, 2000.

[29] A. Haralick, Robert M., Shanmugam. K and I. Dinstein, “TexturalFeatures,” *IEEE Trans. Syst. Man Cybern.*, vol. SMC-3, No., pp. 610–621, 1973.

[30] L. P. Coelho, “Mahotas,” *J. Open Res. Softw.*, 2013.

[31] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, 2005, vol. 1, pp. 886–893 vol. 1.

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

[33] T. Ojala, M. Pietikäinen, and D. Harwood, “A comparative study of texture measures with classification based on featured distributions,” *Pattern Recognit.*, vol. 29, no. 1, pp. 51–59, 1996.

[34] X. Wang, T. X. Han, and S. Yan, “An HOG-LBP human detector with partial occlusion handling,” in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 32–39.

[35] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, no. February, 2001.

[36] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in {P}ython,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[37] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[38] J.-I. Hwang and H.-S. Jung, “Automatic Ship Detection Using the Artificial Neural Network and Support Vector Machine from X-Band Sar Satellite Images,” *Remote Sens.*, vol. 10, no.

11, p. 1799, 2018.

[39] C. Dong, J. Liu, and F. Xu, “Ship detection in optical remote sensing images based on saliency and a rotation-invariant descriptor,” *Remote Sens.*, vol. 10, no. 3, 2018.

[40] J. Cramer, “The Origins of Logistic Regression,” 2002.

[41] J. Tang, C. Deng, G. Bin Huang, and B. Zhao, “Compressed-domain ship detection on spaceborne optical image using deep neural network and extreme learning machine,” *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 3, pp. 1174–1185, 2015.

[42] T. K. Ho, “Random decision forests,” in *Proceedings of 3rd international conference on document analysis and recognition*, 1995, vol. 1, pp. 278–282.

[43] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[44] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.

[45] N. S. Altman, “An introduction to kernel and nearest-neighbor nonparametric regression,” *Am. Stat.*, vol. 46, no. 3, pp. 175–185, 1992.

[46] S. Huang, H. Xu, and X. Xia, “A remote sensing ship recognition using random forest,” *Proc. Sci.*, vol. 18-19-Dece, pp. 1–11, 2015.

[47] L. Breiman, “Arcing the edge,” *Statistics (Ber).*, vol. 4, pp. 1–14, 1997.

[48] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. 13-17-Augu, pp. 785–794, 2016.

[49] L. Breiman, “Bagging predictors: Technical Report No. 421,” *Dep. Stat. Univ. Calif.*, no. 2, p. 19, 1994.

Appendix

A list of the 7 Hu Moments and their formulae:

$hu[0] = n_{20} + n_{02}$

$hu[1] = (n_{20} - n_{02})^2 + 4{n_{11}}^2$

$hu[2] = (n_{30} - 3n_{12})^2 + (3n_{21} - n_{03})^2$

$hu[3] = (n_{30} + n_{12})^2 + (n_{21} + n_{03})^2$

$hu[4] = (n_{30} - 3n_{12})(n_{30} + n_{12})[(n_{30} + n_{12})^2 - 3(n_{21} + n_{03})^2] + (3n_{21} - n_{03})(n_{21} + n_{03})[3(n_{30} + n_{12})^2 - (n_{21} + n_{03})^2]$

$hu[5] = (n_{20} - n_{02})[(n_{30} + n_{12})^2 - (n_{21} + n_{03})^2] + 4n_{11}(n_{30} + n_{12})(n_{21} + n_{03})$

$hu[0] = (3n_{21} - n_{03})(n_{21} + n_{03})[3(n_{30} + n_{12})^2 - (n_{21} + n_{03})^2] - (n_{30} - 3n_{12})(n_{21} + n_{03})[3(n_{30} + n_{12})^2 - (n_{21} + n_{03})^2]$

$where\ n_{ji}\ stands\ for\ moments :: nu_{ji}$

A list of Haralick features and their formulae:

$p(i,j)$ - (i,j)th entry in a normalized gray-tone spatial dependence matrix, = $P(i,j)/R$.

$p_x(i)$ - ith entry in the marginal-probability matrix obtained by summing the rows of $p(i,j), = \sum_{j=1}^{N_g} P(i,j)$.

Ng – Number of distinct gray levels in the quantized image.

$p_y(j) = \sum_{i=1}^{N_g} p(i,j)$

$p_{x+y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \delta_{i+j,k} p(i,j)$ where k = 2,3,….,$2N_g$.

$p_{x-y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} \delta_{|i-j|,k} p(i,j)$ where k = 0,1,2,3,….,$N_g - 1$ and the Kronecker delta function

$\delta_{m,n}$is defined by $\delta_{m,n} = \begin{cases} 1 \; when \; m = n \\ 0 \; when \; m \neq n \end{cases}$

Angular Second Moment: $f_1 = \sum_i \sum_j \{p(i,j)\}^2$

Contrast: $f_2 = \sum_{n=0}^{N_g - 1} n^2 \{\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)\}$, *where |i-j|=n*

Correlation: $f_3 = \frac{\sum_i \sum_j (ij) p(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y}$

Sum of Squares (Variance): $f_4 = \sum_i \sum_j (i - \mu)^2 p(i,j)$

Inverse Difference Moment: $f_5 = \sum_i \sum_j \frac{1}{1+(i-j)^2} p(i,j)$

Sum Average: $f_6 = \sum_{i=2}^{2N_g} i p_{x+y}(i)$

Sum Variance: $f_7 = \sum_{i=2}^{2N_g} (i - f_8)^2 p_{x+y}(i)$

Sum Entropy: $f_8 = -\sum_{i=2}^{2N_g} p_{x+y}(i) \log \{p_{x+y}(i)\}$

Entropy: $f_9 = -\sum_i \sum_j p(i,j) \log (p(i,j))$

Difference Variance: $f_{10} = variance \; of \; p_{x-y}$

Difference Entropy: $f_{11} = -\sum_{i=0}^{N_g - 1} p_{x-y}(i) \log \{p_{x-y}(i)\}$

Information measures of Correlation 1: $f_{12} = \frac{HXY - HXY1}{\max \{HX, HY\}}$

Information measures of Correlation 2: $f_{13} = (1 - \exp\,[\,-2.0\,(HXY2 - HXY)])^2$, where $HXY = -\sum_i \sum_j p(i,j) \log\,(p(i,j))$

Vita

The author, Sylvia Charchut, was born in Chicago, IL. She obtained her bachelor's degree in 2013 from Southeastern Louisiana University in Hammond, LA. She joined the University of New Orleans Computer Science Master's program in Fall 2014. Sylvia has been working full-time as a computer scientist at the Naval Research Laboratory while pursuing her graduate degree.