저작자표시-비영리-변경금지 2.0 대한민국

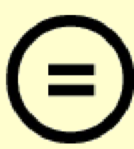
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

공학석사학위논문

기계학습 모형을 이용한 동영상 내 군함 검출

2017년 6월

서울대학교 대학원
산 업 공 학 과
조 창 성

기계학습 모형을 이용한 동영상 내 균함 검출

지도교수 박 종 헌

이 논문을 공학석사학위논문으로 제출함

2017년 6월

서울대학교 대학원

산 업 공 학 과 정 보 경 영

조 창 성

조창성의 석사학위논문을 인준함

2017년 6월

위 원 장 박 용 태 (인)

부위원장 박 종 헌 (인)

위 원 조 성 준 (인)

초 록

현대생활은 인공지능 시대에 살고 있다고 할 수 있다. 그 중 하나로 많은 장소에 카메라를 설치하여 얼굴과 차량번호 등을 자동으로 인식한다. 군에서도 줄어드는 병력을 대신하여 곳곳에 감시카메라를 설치함으로써 경계임무를 수행하고 있다. 해군도 마찬가지로 각종 도서 및 울타리에 감시카메라를 설치하여 해상에 있는 접촉물을 식별한다. 해상 접촉물 중에서 군함을 자동으로 식별해 낸다면 경계임무를 수행함에 있어 많은 도움이 될 것이다. 동영상에서 군함을 검출하는 방법으로 기계학습 모형인 Support Vector Machine과 Random Forest 알고리즘을 사용하였다.

동영상은 실시간 캡처를 하여 이미지로 저장을 할 수 있다. 그러면 이미지에 군함의 여부를 판별하면 될 것이다. 해상 접촉물에는 상선, 여객선, 어선, 군함, 부유물 등이 있다. 이중에서 군함을 식별해 내야 한다. 각 물체들에는 독특한 에지 분포 정보를 가지고 있으며 선박 중의 하나인 군함도 마찬가지이다. 에지 분포 정보를 바탕으로 HOG(Histogram of Oriented Gradient)를 이용한 SVM 실험결과 최대 62.7%로 매우 낮았다. Random Forest 알고리즘은 68.7%로 SVM 알고리즘 보다 6% 더 높게 나왔다.

군함의 선체 색깔은 거의 회색이다. 반면 상선, 여객선, 어선들의 선체는 회색을 사용하지 않고 흰색, 적색, 청색 계통의 색깔을 주로 사용한다. 그래서 HOG 뿐만이 아니라 색상 정보를 SVM의 입력자료로 이용하였다. 실험결과 최대 77.2%의 정해률이 나왔고, 정밀도도 80.6%로 다른 알고리즘보다 상대적으로 높았다. 그러나 SVM의 알고리즘은 결과 값을 산출하는데 상대적으로 긴 시간이 소요되었다.

참 긍정율은 Random Forest에서 최대 81.9%의 값을 보였다. 그러나 정밀도는 69.1%로 낮게 나왔다.

군대에서 경계는 매우 중요하다. 경계의 한 번의 실수가 작전의 승패를 좌지우지 하는 경우가 많이 있다. 군대에서는 경계를 함에 있어서 반드시 100%를 달성해야 한다. SVM의 참 긍정율 71.3%, Random Forest의 참 긍정율 81.9%는 100%에 비해 상당히 낮은 수치이다. 또한 경계임무에 있어서 71.3%, 81.9%는 용납될 수 없다. 하지만 군함 검출 시스템을 병행해서 사용한다면 경계임무에 도움이 될 것이라 생각한다.

주요어 : 기계학습, 검출, 군함, SVM, Support Vector Machine, Random Forest, HOG

학 번 : 2007 - 20652

목 차

1. 서 론 ········· 1
1.1. 연구목적 ········· 1
1.2. 논문소개 ········· 2
1.3. 논문구성 ········· 3
2. 배경 이론 및 관련 연구 ········· 4
2.1. Support Vector Machine ········· 4
2.1.1. Margin method ········· 6
2.1.2. Kernel method ········· 8
2.2. Random Forest ········· 10
2.3. 기계학습 및 객체 검출 ········· 10
2.4. HOG ········· 13
2.4.1. HOG(Histogram of Oriented Gradient) ········· 13
2.4.2. 에지 분포를 이용한 HOG ········· 15
2.5. RGB 색상 ········· 20
2.6. 선박의 종류 및 특징 ········· 21
2.6.1. 선박의 종류 ········· 21
2.6.2. 선박 종류별 특징 ········· 22
3. 이미지의 HOG 및 색상 정보 ········· 23
3.1. HOG Feature ········· 24
3.2. HOG를 위한 에지 정보 검출 ········· 27
3.3. 이미지 내 색상 검출 ········· 28
3.3.1. 회색 및 흰색 검출 ········· 28
3.3.2. 적색 및 청색 검출 ········· 30

4. 실험 및 평가 ··· 32
4.1. 실험 데이터 ··· 32
4.2. 실험환경 ··· 35
4.3. 평가지표 ··· 35
4.4. 실험결과 ··· 38
4.4.1. 정해율(Accuracy) ··· 38
4.4.1.1. HOG만을 이용한 알고리즘 평가 ··· 38
4.4.1.2. HOG와 색상을 이용한 알고리즘 평가 ··· 38
4.4.2. 참 긍정율(Recall rate) ··· 47
4.4.3. 정밀도(Precision) ··· 49
4.4.4. 종합 ··· 51
5. 결론 ··· 52
5.1. 요약 및 의의 ··· 52
5.2. 후속 과제 ··· 54
참고문헌 ··· 56
Abstract ··· 59
Appendix A(Main.java) ··· 61
Appendix B(MyClassifier.java) ··· 65

표 목차

[표 2-1] RGB 색상표 ···· 20
[표 2-2] 선종분류표 ···· 21
[표 2-3] 선박 종류별 특징 ···· 22
[표 3-1] HOG Cell 크기에 따른 Cell 수 ···· 26
[표 4-1] 혼동행렬(Confusion Matrix) ···· 36
[표 4-2] 정해률, 참 긍정률, 정밀도의 가장 큰 값 ···· 51
[표 4-3] 정해률, 참 긍정률, 정밀도의 가장 큰 값의 혼동행렬 ···· 51

그림 목차

[그림 2-1] SVM의 개념 ······ 4
[그림 2-2] 고차원으로의 변환 ······ 8
[그림 3-1] 실험과정 ······ 23
[그림 3-2] 알고리즘 ······ 23
[그림 3-3] HOG Feature 과정 ······ 25
[그림 3-4] Filter에 따른 에지 분포 이미지 ······ 27
[그림 3-5] 이미지의 흰색 값을 구하기 위한 슈도 코드 ······ 28
[그림 3-6] 이미지의 흰색 값을 구하기 위한 슈도 코드 ······ 29
[그림 3-7] 이미지 내 회색 및 흰색 검출 ······ 29
[그림 3-8] 적색의 색상 범위 ······ 30
[그림 3-9] 청색의 색상 범위 ······ 30
[그림 3-10] 이미지의 적색 값을 구하기 위한 슈도 코드 ······ 30
[그림 3-11] 이미지의 청색 값을 구하기 위한 슈도 코드 ······ 31
[그림 3-12] 이미지 내 적색 및 청색 검출 ······ 31
[그림 4-1] 군 함 ······ 33
[그림 4-2] 상선, 여객선, 어선 ······ 34
[그림 4-3] HOG만을 이용한 filter별 알고리즘 정해률 ······ 38
[그림 4-4] HOG만을 이용한 filter별 알고리즘 소요 시간 ······ 39
[그림 4-5] HOG Cell 3×3, Prewitt filter ······ 40
[그림 4-6] HOG Cell 3×3, Prewitt filter ······ 41
[그림 4-7] HOG Cell 6×6, Prewitt filter ······ 41
[그림 4-8] HOG Cell 3×3, Robert filter ······ 42
[그림 4-9] HOG Cell 4×4, Robert filter ······ 42

[그림 4-10]　HOG Cell 6×6, Robert filter ·· 43
[그림 4-11]　HOG Cell 3×3, Sobel filter ·· 43
[그림 4-12]　HOG Cell 4×4, Sobel filter ·· 44
[그림 4-13]　HOG Cell 6×6, Sobel filter ·· 44
[그림 4-14]　HOG Cell 3×3, LoG filter ·· 45
[그림 4-15]　HOG Cell 4×4, LoG filter ·· 45
[그림 4-16]　HOG Cell 6×6, LoG filter ·· 46
[그림 4-17] HOG Cell 3×3에 대한 참 긍정율 ·· 47
[그림 4-18] HOG Cell 4×4에 대한 참 긍정율 ·· 48
[그림 4-19] HOG Cell 6×6에 대한 참 긍정율 ·· 48
[그림 4-20] HOG Cell 3×3에 대한 정밀도 ·· 49
[그림 4-21] HOG Cell 4×4에 대한 정밀도 ·· 50
[그림 4-22] HOG Cell 6×6에 대한 정밀도 ·· 50
[그림 5-1] 본 논문에서의 실험방법 ·· 54
[그림 5-2] 제안하는 실험방법 ·· 54

1. 서 론

1.1. 연구목적

현대생활은 인공지능 시대에 살고 있다고 할 수 있다. 인간의 지능으로 할 수 있는 사고, 학습, 자기 개발 등을 컴퓨터가 할 수 있도록 하는 방법을 연구하는 컴퓨터 공학 및 정보기술의 한 분야로서, 컴퓨터가 인간의 지능적인 행동을 모방할 수 있도록 하는 것을 인공지능이라고 말하고 있다.[1] 그런 분야 중의 하나로 컴퓨터가 TV 카메라를 통해 잡은 영상을 분석하여 그것이 무엇인지를 알아내거나, 사람의 목소리를 듣고 그것을 문장으로 변환하는 것 등이 있다. 예를 들면 주차장에 들어갈 때 카메라를 통해 차량 번호판을 자동으로 인식하는 것이다.

감시카메라는 많은 곳에서 사용되고 있다. 감시카메라는 사람들이 움직이고 차량이 이동하는 곳 등 거의 모든 구역에 설치되어 많은 상황들이 실시간 녹화되고 있다. 녹화된 자료는 범인을 찾을 때 이용하는 등 실용적으로 많이 사용된다.

우리나라의 인구는 점점 줄어들고 있다. 인구감소는 군에도 영향을 미쳐 군의 정원이 줄어들고 있다. 그래서 군에서는 예전에 사람이 배치되어 감시한 곳을 카메라를 설치하여 경계와 감시를 대신한다. 그것은 해군도 마찬가지이다. 과거에는 도서에 많은 인원을 배치하여 해상을 감시하였으나 지금은 레이다와 카메라를 도서에 설치하여 원격으로 경계 및 감시를 실시한다. 그러다보니 때로는 한 사람이 여러 개의 감시카메

---

1) 네이버 두산백과, 인공지능, http://terms.naver.com/entry.nhn?docId=1136027&cid=40942&categoryId=32845 (2017. 6. 1.)

라를 볼 때도 있다. 해군은 기본적으로 레이다를 이용하여 해상접촉물을 접촉하고 식별하지만 때로는 감시카메라를 이용하여 해상접촉물을 감시하고 최종식별을 한다. 특히 접적해역에서는 적 함정을 식별하는데 많이 이용되고 있다. 감시카메라에 촬영된 접촉물 중에서 함정(군함)을 구별해낸다면 경계임무에 많은 도움이 될 것이다. CCTV 및 고성능카메라에 접촉된 선박 중 북한함정을 식별하여 경보를 해 준다면 감시자 및 작동수가 쉽게 북한함정을 감시할 수 있을 것이다.

1.2. 논문소개

본 논문은 카메라에 촬영된 접촉물을 식별하기 위해서 영상을 주기적(1초 단위 등)으로 화면캡처를 하여 이미지로 저장하고 이미지에 함정이 있는가 없는가를 판별하여 동영상 내 군함을 검출해 내는 것을 목표로 한다. 이미지 내에 군함을 검출하기 위해 기계학습 모형인 Support Vector Machine(SVM)과 Random Forest를 이용하였다. 실험의 입력자료로는 Histogram of Oriented Gradient(HOG)와 이미지의 색상(회색, 적색, 흰색, 청색)이 있다. 예를 들면 군함은 대부분 회색이므로 이미지에 회색이 많다면 군함이 포함될 가능성이 높을 것이다. HOG를 위한 이미지의 에지 분포 정보를 검출하기 위해 Prewitt filter, Rober filter, Sobel filter, LoG filter등을 이용하였다. 또한 알고리즘의 성능평가를 위해 SVM, Random Forest, Naive Bayes, J48 알고리즘의 실험도 병행하였다.

1.3. 논문구성

본 논문은 총 5장으로 구성 되어 있다. 본 논문에서는 이미지의 에지 분포 및 색상정보를 바탕으로 SVM, Random Forest 실험을 하여 군함을 검출하는 것을 목적으로 한다.

2장 ‘배경이론 및 관련연구’에서는 먼저 논문과 관련된 SVM, Random Forest, HOG, RGB 색상, 기계학습에 대해 기술하였다. 그리고 선박의 분류와 군함이 다른 선박들과 어떻게 비교되는 가를 설명하였다.

3장 ‘이미지의 HOG 및 색상정보’에서는 실험과정 및 알고리즘을 설명하고 HOG Feature 및 이미지에서 어떻게 색상 정보를 검출하는 가에 대해 설명하였다.

4장 ‘실험 및 결과’에서는 HOG 및 색상 정보를 이용한 SVM과 Random Forest의 결과를 산출하였다. 이때 실험의 비교분석을 위해 Naive Bayes, J48, SVM, Random Forest 알고리즘도 실험을 하였다. 평가지표는 정해률(Accuracy), 참 긍정율(Recall rate), 정밀도(Precision)로 실시하였다.

5장 ‘결론’에서는 본 논문을 요약하고 후속과제를 제시하였다.

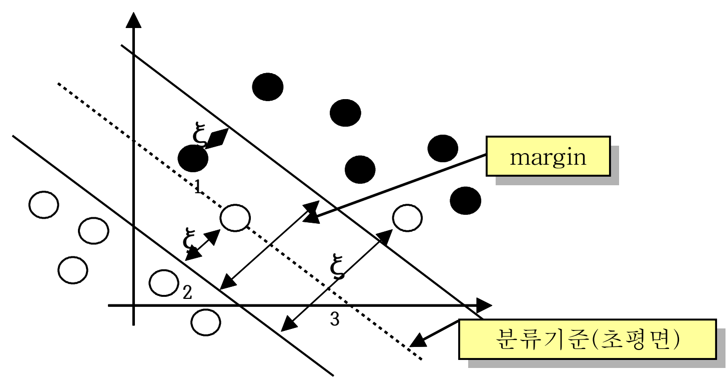
2. 배경 이론 및 관련 연구

2.1. Support Vector Machine

Support Vector Machine(SVM)은 2개의 범주를 분류하는 이진 분류기의 한 종류이다. SVM은 이항 반응 변수의 두 벡터를 분리할 때 각 그룹에 속한 벡터 중 초평면(hyperplane)까지의 거리가 가장 가까운 벡터들이 먼 거리를 가지도록, 즉 Margin을 최대화시키는 두 개의 그룹으로 분류하며 [그림 2-1]과 같다. 이때, 다른 집단의 영역에 포함되는 벡터들을 고려하여 Slack variable을 생성한다. 또한 선형 분류가 힘들 때 kernel 함수 기법으로 고차원의 선형 경계를 만들어 저차원의 비선형 경계를 생성한다.[6]

먼저 선형분리문제에 대해 생각해보면, 하나의 집합과 다른 집합을 세밀하게 분류시킬 수 있는 최적의 분리 경계면을 찾아주는 것이다. 이러한 최적 분리 경계면은 훈련 벡터뿐 아니라 각 집합 내 숨어있는 벡터들 또한 분류함을 말한다. 최적의 경계면, 즉 두 집합 사이를 통과하는 경계면을 초평면으로 정의한다. 그리고 경계면 가장 가까이에 있는 벡터들을 support vector라고 한다.[6]

x, y를 자료 공간 내 N개의 벡터라 두자.



[그림 2-1] SVM의 개념

$$(X_1, Y_1), (X_2, Y_2), \bullet \bullet \bullet, (X_N, Y_N)$$

$$X_i \in R^P \text{ and } y_i \in -1, 1$$

분리경계면의 하나로서 초평면은 식 2-1과 같이 정의된다.

$$\left\{ x : f(x) = x^T\beta + \beta_0 = 0 \right\} \quad (2\text{-}1)$$

여기서 β는 weight coefficient vector이고, β_0은 bias 항이다. ‖β‖ = 1이고 분류기준은 이 초평면을 중심으로, $G(x) = sign(x^T\beta + \beta_0)$이다.

제약조건 $y_i(x^t\beta + \beta_0) > M, i = 1, \bullet \bullet \bullet, N$ 에 대하여 $\max_{\beta, \beta_0} C$

Support vector와 최적 초평면의 거리를 C = 1/‖β‖ 로 나타낼 수 있다. 훈련벡터 X_i와 초평면까지의 수직거리중 가장 큰 값을 margin이라고 하며 다음과 같이 훈련자료들의 마진 C > 0를 최대화하는 유일한 초평면을 찾아 준다. 즉, SVM의 해를 구하는 것은 식 2-2와 같은 Cell 최적화 문제가 된다.

$$minimize \ \beta^t\beta \quad (2\text{-}2)$$

$$subject \ to \ y_i(xt\beta + \beta_0) \geq 1$$

2.1.1. Margin method

위에서 언급한 식은 선형분리 가능한 집합에 대해서만 적용가능하다. 그러나 이러한 벡터들을 완벽하게 분리하는 초평면을 찾기란 드물며 따라서 다른 집단의 영역에 포함되는 벡터들을 고려하여 Slack variable ξ_1, ξ_2, …, ξ_n을 생성한다. 이 방법은 제약조건을 식 2-3과 같이 놓는다.[18]

$$\min_{\beta,\beta_0} \| \beta \| \tag{2-3}$$

$$s.t\ \forall_i,\ y_i(x T_i \beta + \beta_0) \geq 1 - \xi_i$$

$$\xi_i \geq 0,\ \sum_{i=1}^{N} \xi_i \leq constant$$

ξ에 대한 제약식을 계산의 편의를 위해 다시 표현하면 합의 형태로 표현이 가능하다.

$$\min_{\beta,\beta_0} \frac{1}{2} \| \beta \|^2 + c \sum_{i=1}^{N} \xi_i \tag{2-4}$$

$$s.t\ \ y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i\ \ \forall_i$$

C는 tuning parameter의 역할을 한다. C가 작아질수록 고려되는 점(Support Vector)의 개수를 많이 잡게 된다. 라그랑지 승수법을 사용하여 βi, β0, ξi에 대해 문제를 풀면,

$$L_p = \frac{1}{2} \| \beta \|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{n} \alpha_i [y_i(x_i \beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N} u_i \xi_i \tag{2-5}$$

$$\hat{\beta}=\sum_{i=1}^{N}\alpha_i y_i x_i$$

$$0=\alpha_i y_i$$

$$\hat{\alpha}=C-u_i \forall_i$$

대입하여 식 2-6과 같이 나타낼 수 있다.

$$L_D=\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i=1}^{N}\alpha_i\alpha'_i y_i y'_i x_i^T x'_i \qquad (2\text{-}6)$$

$$0 \le \alpha_i \le C$$

$$\sum_{i=1}^{N}\alpha_i y_i=0$$

Kuhn-Tucker theorem에 따르면 추정하려는 변수들 β_i, β_0, ξ_i로 이루어진 공간의 안장점에서 식 2-7과 같이 라그랑지 승수와 조건식의 관계가 나타난다.

$$\alpha_i\left[y_i(xT_i\beta+\beta_o)-(1-\xi_i)\right]=0 \qquad (2\text{-}7)$$

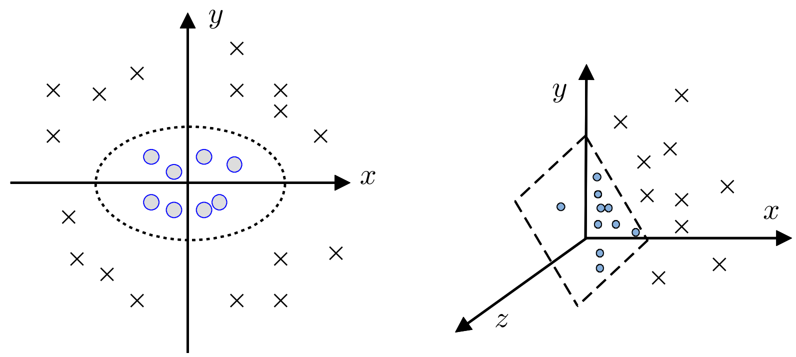
$$u_i\xi_i=0$$

$$y_i(xT_i\beta+\beta_0)-(1-\xi_i) \ge 0$$

위 조건식을 만족하는 값들은 앞에서 본 ξ_i와 Support Vector의 관계를 만족하고, 유일한 해를 구하는 근거가 된다. 이제 Ld는 일반적 Convex Quadratic Programming으로 풀 수 있다.

2.1.2. Kernel method

Margin은 선형분리의 확장이라 볼 수 있다. 그런데 대부분의 패턴은 선형적으로 분리가 가능하지 않다. 따라서 비선형 패턴을 분리하기 위하여 비선형 패턴의 입력공간을 선형 패턴의 feature space로 변환한다. Kernel method는 진정한 비선형 경계면을 찾는 방법이다. 다음 아래의 [그림 2-2]는 선형 분리되지 않는 예이다.[18]



[그림 2-2] 고차원으로의 변환

2차원 공간이 3차원으로 변환된다면 서로 다른 두 class vector는 선형적으로 분리되는 것을 볼 수 있다. 입력벡터 x_i 의 고차공간으로의 변환을 $h(x_i)$ 라고 하자.

$$K(x, x') = \, < h(x), h(x') >$$

feature space 내에서 분리경계면은 식 2-8과 같다.

$$\hat{f}(x) = x^T\hat{\beta} + \hat{\beta_0},\ \hat{\beta} = \sum_{i=1}^{N}\alpha_i y_i x_i \qquad (2\text{-}8)$$

$$\Rightarrow \hat{f}(x) = \sum_{i=1}^{N}\alpha_i y_i < x, x_i > + \hat{\beta_0}$$

기저의 변환에 따라 분류함수의 해는

$$\hat{f}(x) = \sum_{i=1}^{N}\alpha_i y_i < h(x), h(x_i) > + \hat{\beta_0} \qquad (2\text{-}9)$$

$$\hat{f}(x) = \sum_{i=1}^{N} v_i < h(x),\ h(x_i) > + \hat{\beta_0} \qquad (2\text{-}10)$$

로 나타낼 수 있다.

Kernel 함수의 예로는 식 2-11, 2-12와 같다.

d차 다항식 : $(1 + < x, x' >)^d$ (2-11)

방사형 기저(Radial basis) : $\exp(-d \| x - x' \|^2$ (2-12)

앞에서 구했던 SVM 최적화 조건식은 식 2-13과 같다.

$$\min_{\beta, \beta_0} \frac{1}{2} \| \beta \|^2 + C\sum_{i=1}^{N}\xi_i \qquad (2\text{-}13)$$

$$s.t\ \ y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i\ \ \forall_i$$

위 식의 두 번째 항은 misclassification에 대한 penalty 항이고, 상수 C는 두 번째 항의 degree이다.

2.2. Random Forest

Random Forest는 무작위로 추출한 자료들을 이용하여 많은 수의 의사결정나무를 만들고 생성된 여러 의사결정나무의 식별 클래스들을 가중표시하여 최종 클래스를 판단하는 분류 방법이다. Random Forest의 좋은 점은 많은 입력 자료들을 처리할 수 있다. 또한 부트스트랩(bootstrap) 기법을 사용하여 훈련 자료의 생성을 통한 상당 수의 의사결정나무를 만들기 때문에 다양한 양식에 대해 정확성이 높다.

Random Forest에서 margin은 식 2-14와 같이 정의한다.

$$margin(X, Y) = P(\widehat{Y}_{\theta} = Y) - \max_{L \neq Y} P(\widehat{Y}_{\theta} = Z) \qquad (2\text{-}14)$$

여기서 $\widehat{Y}_{\theta}$는 무작위 벡터 $\hat{\theta}$로부터 세워진 분류기에 의하여 예측된 X의 클래스이다. margin 값이 크다는 것은 분류기가 X를 더 잘 예측할 가능성이 높다는 것이다. 나무의 수가 충분히 클 경우 Random Forest의 일반화 오류의 상한선(risk)은 식 2-15와 같이 수렴한다.

$$risk \leq \frac{\bar{p}(1-s^2)}{s^2} \qquad (2\text{-}15)$$

식 2-15에서 $\bar{p}$는 나무들 사이의 평균 상관관계이며 s는 나무 분류기의 강도(strength)를 나타내는데 분류기 집합의 강도는 분류기들의 평균 margin을 뜻한다. 나무 사이의 상관관계가 커지거나 강도가 줄어드면 일반화 오류의 한계 값은 증가한다. 일반화 오류를 줄이기 위해서 나무사이의 상관관계를 줄여주는 무작위성을 이용할 수 있다.

2.3. 기계학습 및 객체 검출

프로그래머가 손수 많은 규칙을 만들어 알고리즘에 포함시키는 전문가 시스템 기반의 인공지능(AI)은 의학 등의 단순하고 명확한 규칙들을 데이터베이스화하여 사용하는 방법으로 우수한 성능을 보였으나, 음성인식이나 이미지를 이용한 객체 검출과 같이 단순하고 명확한 규칙을 프로그램화하기 곤란한 범위에서는 성능에 한계가 있었다[1][15].

전문가 시스템의 한계를 넘어서기 위한 방법 중의 하나가 기계학습이다. 컴퓨터를 학습시켜 기하학, 통계 또는 안공 신경망을 사용한 추론을 기반으로 스스로 규칙을 만들도록 한다. 기계학습은 "Task(*T*)에 대해 계속되는 경험 '*E*'를 통하여 그 *T*에 대한 성능을 높이는 것"으로 정의한다[28]. 기계학습에서 가장 중요한 것은 '*E*'에 해당하는 자료이다. 좋은 품질의 자료 '*E*'가 많이 있다면 보다 높은 결과를 끌어낼 수 있다[1][4].

기계학습은 학습 방법에 따라서 강화 학습(Reinforcement Learning), 지도 학습(Supervised Learning), 비지도 학습(Unsupervised Learning)으로 나뉘며, 자료의 검출 방법에 따라 확률 기반 알고리즘, 기하학 기반 알고리즘, 인공 신경망 기반 알고리즘으로 나눌 수 있다[1][3].

지도 학습은 프로그래머가 각각의 입력 값에서 분류된 자료를 컴퓨터에 입력하면 컴퓨터가 그것을 학습하는 것이다. 프로그래머가 관여하므로 정확도가 높은 데이터를 이용할 수 있다는 장점이 있는 반면 얻을 수 있는 자료의 양이 적다는 문제점이 있다. 비지도 학습은 프로그래머의 관여 없이 분류되지 않은 자료를 컴퓨터가 학습하는 것으로, 학습이 정확한 지 확인할 방법은 없지만 거의 모든 정보를 자료화 할 수 있는 장점이 있다. 강화 학습은 바둑게임과 같은 특정 상태(State)에서 어떤 행위

(Action)를 할 때 주어지는 대가(Reward)를 최대화 하는 방향으로 학습이 된다[1][19].

데이터의 분류 및 검출 방법에 따른 기계학습 분류 중에서 확률 기반 알고리즘은 "X가 주어졌을 때 Y가 발생할 확률"을 예상한다. 기하학 기반 알고리즘은 주어진 입력 X의 특징(feature)을 벡터(vector)로 만들어 특징 벡터간의 거리와 같은 기하학적인 관계를 기반으로 Y가 발생할 확률을 예측한다.[1][19]

인공 신경망 기반 알고리즘은 가상의 뉴런을 수학적으로 모델링한 후 시뮬레이션 하는 알고리즘이다. 상대적으로 뛰어난 성능을 보이지만, 학습 시간이 길고 연산량과 학습 데이터를 많이 필요로 한다[1][9].

2.4. HOG

2.4.1. HOG(Histogram of Oriented Gradient)

HOG 특징은 이미지에서 물체를 검출하기 위한 컴퓨터의 영상처리 알고리즘에 사용된다. 이 방법은 이미지의 국한된 부분 안에서 기울기 방향의 수를 계산하는 것이며, 밝기, 기울기, 에지 분포에 따라 그 값이 결정된다. 이러한 이미지 내부의 특징들은, 이미지를 셀(cell)이라는 서로 연결된 작은 영역으로 나누고 각 셀 내에 있는 픽셀(pixel)에 대한 기울기 방향, 에지 방향의 히스토그램을 만들어서 나타낸다. 이러한 히스토그램들이 모여서 이미지 내부의 특징을 나타낼 수 있다. 또한 이미지 내부의 특징을 보다 잘 나타내기 위하여 블록(block)이라는 셀보다 큰 이미지 영역에 걸친 명암을 측정하여 블록 내의 모든 셀들을 정규화함으로써 조명이 변화하는 상황에서도 알고리즘이 일정한 성능을 유지 할 수 있게 해준다[1][13].

HOG 과정은 크게 3단계로 나뉜다. 첫 번째 단계는 1 차원 중심점 이산 미분 마스크를 수평/수직 방향으로 적용하여 이미지 내부의 기울기 값을 계산한다[1][13].

$$D_X = [-1 \quad 0 \quad 1], \qquad D_Y = [-1 \quad 0 \quad 1] \tag{2-16}$$

그리고 이미지 I 가 주어졌을 때, 식 2-16의 마스크와 이미지를 각각 x 방향과 y 방향으로 식 2-17과 같이 convolution 연산한다. [1][16]

$$I_X = I * D_X, \qquad I_Y = I * D_Y \tag{2-17}$$

위에서 구한 I_X와 I_Y를 이용하여 이미지의 기울기의 크기와 방향을 다음과 같이 계산한다[1][16].

$$\text{기울기의 크기} : |G| = \sqrt{I_X^2 + I_Y^2} \tag{2-18}$$

$$\text{기울기의 방향} : \theta = \arctan\frac{I_Y}{I_X} \tag{2-19}$$

두 번째 단계에서는 각 셀의 히스토그램은 계산한다. 셀 내에 각 픽셀 값은 상기한 기울기 계산을 통해서 기울기의 방향이 연산이 되고 이 값들은 빈(bin) 수로 설정된 각각의 방향 히스토그램 대역들에 퍼지게 된다. 각 셀들은 이미지 내에서 사각형으로 이루어져 있고, 히스토그램 대역들은 0°에서 360° 또는 0°에서 180°까지 분포한다[1][16].

세 번째 단계에서는 조명변화에 따라 검출 대상의 형상이 변하는 것에 대응하기 위하여 기울기의 크기를 국부적으로 정규화 한다. 이 과정에서 셀들을 더 크고 공간적으로 연결된 블록들로 그룹화 한다[1][12]. 블록을 정규화 하는 방법에는 크게 3가지가 있다. 주어진 블록 내의 모든 히스토그램을 포함하는 비정규화된 벡터를 ν, $\|\nu\|$를 k 놈(norm), e는 상수라고 했을 때 정규화 방법은 식 2-20, 2-21, 2-22와 같다[1][7].

$$L_1 - norm : f = \frac{\nu}{\|\nu\|_1 + e} \tag{2-20}$$

$$L_1 - sqrt : f = \sqrt{\frac{\nu}{\|\nu\|_1 + e}} \tag{2-21}$$

$$L_2 - norm \ : \ f = \frac{\nu}{\sqrt{\| \nu \|_2^2 + e^2}} \quad (2\text{-}22)$$

본 논문에서는 정규화 방법 중에서 식 2-20의 $L_1 - norm$을 사용한다.

2.4.2. 에지 분포를 이용한 HOG

HOG를 하기 위해서는 에지 분포 정보를 이용하는 기법에 주로 사용되는 대표적인 것으로는 Prewitt filter, Sobel filter, Robert filter, LoG filter 등이 있다.

에지란 주어진 문턱치를 넘어서는 화소 값들의 국부적인 불연속으로 정의된다. 즉 영상에서 관측할 수 있는 화소 값의 차이를 갖는 부분이다. 이러한 에지는 이미지에서 유용한 정보를 포함하고 있다. 영상에서 물체를 인식하고 분류하기 위해 사용하는 것 중의 하나가 에지이다[1][10].

에지를 검출하는 데에는 다양한 종류의 filter가 사용된다[1][2]. 에지의 종류는 회색 레벨 값이 천천히 변하는 경사 에지(clamp edge)와 회색 레벨 값이 급격하게 변하는 계단 에지(step edge) 두 가지 형태가 있다. 프로파일을 제공하는 함수를 $f(x)$라고 하면 이것의 도함수는 $f'(x)$이다. 도함수의 값은 화소의 값들이 똑같은 부분에서는 모두 0이 되고, 화소의 값이 차이가 있는 부분에서는 0이 되지 않는다. 대부분의 에지 검출 연산자들은 미분을 바탕으로 한다. 이산 이미지에 대해 도함수의 정의는 식 2-23과 같다[1][14].

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \quad (2\text{-}23)$$

이미지에서 h의 가장 작은 값은 1이다. 그래서 연속적인 도함수 정의를 이산형태로 표현하면 식 2-24와 같다[1][14].

$$f(x+1)-f(x) \tag{2-24}$$

이 도함수에 대한 다른 표현은 식 2-25, 2-26과 같으며,

$$\lim_{h\to 0}\frac{f(x)-f(x-h)}{h} \tag{2-25}$$

$$\lim_{h\to 0}\frac{f(x+h)-f(x-h)}{2h} \tag{2-26}$$

이의 이산형태는 식 2-27, 2-28과 같다.

$$f(x)-f(x-1) \tag{2-27}$$

$$(f(x+1)-f(x-1)) \tag{2-28}$$

2차원의 이미지에 대해서는 편미분(partial derivatives)을 사용하며, 기울기(gradient)는 식 2-29와 같다.

$$\left[\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\right] \tag{2-29}$$

도함수는 함수 $f(x,y)$에 대해서 가장 크게 증가하는 방향의 벡터이다. 이 벡터의 증가방향은 식 2-30과 같이 각도로 표시되고,

$$\tan^{-1}\left(\frac{\partial f/\partial y}{\partial f/\partial x}\right) \tag{2-30}$$

그 크기는 식 2-31과 같다.

$$\sqrt{\left(\frac{\partial f}{\partial x}\right)^2+\left(\frac{\partial f}{\partial y}\right)^2} \tag{2-31}$$

대부분의 에지 검출 방법은 기울기의 크기를 구하고, 이 결과에 문턱치 처리를 적용한다.

위의 도함수에 대해서 $f(x+1)-f(x)$ 표현을 사용하고 스케일링(Scaling) 요소를 무시하면 수평과 수직방향의 filter는 식 2-32과 같다.

$$[-1\ 0\ 1] \quad \text{and} \quad \begin{bmatrix}-1\\0\\1\end{bmatrix} \tag{2-32}$$

이 filter 는 영상에서 각각 수직 에지와 수평 에지를 찾는다. 영상에서 에지가 갑자기 변화하는 경우에는 아래의 filter를 사용하여 반대방향으로 스무딩(smoothing) 처리를 할 수 있다[1][14].

$$[\ 1\ \ 1\ \ 1\ ] \quad \text{and} \quad \begin{bmatrix}1\\1\\1\end{bmatrix} \tag{2-33}$$

이 두 filter 를 동시에 적용하여 결합된 filter는 식 2-34와 같다[1][14].

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{2-34}$$

이 filter 와 다음과 같이 수평 에지를 검출하는 동반 filter를 에지 검출을 위한 Prewitt filter 라고 한다[1][15].

$$P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2-35}$$

p_x와 p_y가 영상에 P_x와 P_y를 적용하여 얻은 Gray level 값들이면, 그 기울기의 크기는 식 2-36, 2-37, 2-38과 같이 얻을 수 있다[1][20].

$$\sqrt{p_x^2 + p_y^2} \tag{2-36}$$

또는

$$\max|p_x|, |p_y| \tag{2-37}$$

또는

$$|p_x| + |p_y| \tag{2-38}$$

이와 유사한 에지 검출 filter 로 식 2-39의 Robert filter와 식 2-40의 Sobel filter가 있다[20]. Sobel filter는 Prewitt filter와 유사하지만 스무싱은 [1 2 1] 형태를 취한다. 이는 가운데 화소에 더 많은 비중을 두도록 만든다[1][17].

$$\text{Robert filter} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (2\text{-}39)$$

$$\text{Sobel filter} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (2\text{-}40)$$

이와는 다른 방법으로 에지를 검출하는 방법은 2차 도함수를 사용하는 것이다. 양방향의 2차 도함수의 합을 Laplacian이라고 한다. 이는 아래와 같이 구현되고 Laplacian filter로 구현 될 수 있다[1][5].

영상을 식 2-41과 같이 Gaussian filter로 스무딩 처리한 후 그 결과를 식 2-42와 같이 Laplacian filter로 회선처리 하는 것을 LoG(Laplacian of Gaussian) filter 라 한다[1][5].

$$\text{Gaussian filter} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ 또는 } \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (2\text{-}41)$$

$$\text{Laplacian filter} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{ 또는 } \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix} \quad (2\text{-}42)$$

2.5. RGB 색상

RGB 색공간은 컬러 TV나 모니터 등 출력장치가 아닌 빛을 이용하는 디지털 장치에서 사용하는 디지털 색공간이다. 빛의 3원색인 빨강(R), 초록(G), 파랑(B)을 기본으로 빛을 혼합하여 색을 표현하며 24비트 컬러 모니터의 경우에 세 가지 색상을 0에서 255까지의 범위 중에 한 가지 숫자씩 입력하면 색이 만들어진다. 빛의 3원색의 밝기에 따라 각각 0~255까지 숫자를 사용하여 좌표(R, G, B)로 표현한다. [표 2-1]과 같이 (0, 0, 0)은 검은색, (255,255,255)는 흰색, (255, 0, 0)은 빨간색, (0,255, 0)은 초록색, (0, 0, 255)는 파란색이다. 그리고 (100,100,100)과 같이 입력된 세 개의 값이 같은 수일 경우는 회색을 나타낸다.

[표 2-1] RGB 색상표

구 분	R	G	B	색상	구 분	R	G	B	색상
적 색	255	0	0		회색1	220	220	220	
녹 색	0	255	0		회색2	160	160	160	
청 색	0		255		회색3	100	100	100	
검정색	0	0	0		회색4	40	40	40	
흰 색	255	255	255		회색5	210	200	195	
노 랑	255	255	0		회색6	150	140	155	
시안	0	255	255		회색7	100	80	95	
마젠타	255	0	255		회색8	50	65	55	

2.6. 선박의 종류 및 특징

2.6.1. 선박의 종류

선박은 사용목적에 따라 상선, 함정, 어선, 특수작업선으로 크게 구분할 수 있다. 상선은 여객 또는 화물을 운반하여 운임수입을 얻는 것을 목적으로 하는 선박을 말하며, 이것을 다시 화물선, 화객선, 여객선으로 구분할 수 있으며 [표 2-2]의 선종분류표[2]와 같다.

[표 2-2] 선종분류표

<table>
<tr><th colspan="2">구 분</th><th colspan="2">중분류 및 세분류</th></tr>
<tr><td rowspan="11">상
선</td><td rowspan="4">탱커</td><td>원유운반선</td><td>원유</td></tr>
<tr><td>정유운반선</td><td>휘발유, 경유, 중유 등</td></tr>
<tr><td>화학제품운반선</td><td>Sulphur, Naphtha 등</td></tr>
<tr><td>가스운반선</td><td>LPG, LNG</td></tr>
<tr><td>겸용선</td><td>Combined Carrier</td><td>Ore/Bulk/Oil, Ore/Oil, Oil/Bulk 등</td></tr>
<tr><td rowspan="6">건
화
물
선</td><td>Bulk Carrier</td><td>Ore, Coal, Grain, Cement, Log</td></tr>
<tr><td>General Cargo Carrier</td><td>Lumber 등</td></tr>
<tr><td>Full Container Ship</td><td>Container 이외의 포장화물
Container</td></tr>
<tr><td>Pure Car Carrier</td><td>각종 차량 등</td></tr>
<tr><td>Multi Purpose Cargo Carrier</td><td>General Cargo/Bulk/Container</td></tr>
<tr><td>Reefer</td><td>냉장 및 냉동화물</td></tr>
<tr><td colspan="2">어 선</td><td colspan="2">어로선, 공선, 모선, 운반어선, Trawler, Stern Trawler, 참치선망어선, 유자망어선, 포경선, 어업지도선, 어업조사선</td></tr>
<tr><td colspan="2">특수작업선</td><td colspan="2">수로측량선, 해양관측선, 해저전선부설선, 공작선, 기중기선, Tug Boat, Supply Vessel, 소방선, 해양오염방제선, 병원선</td></tr>
<tr><td colspan="2">함 정</td><td colspan="2">전투함, 순양전함, 순양함, 경순양함, 구축함, 잠수함, 원자력잠수함, 항공모함, 소해정, LST, LSM</td></tr>
</table>

2) 출처 : 대한민국 조선협회

2.6.2. 선박 종류별 특징

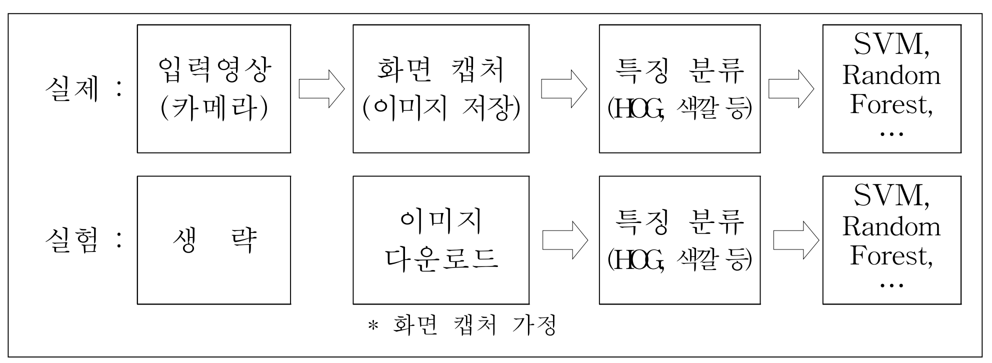
선박 종류별 특징은 [표 2-3]과 같다. 군함의 선체 색깔은 회색이며, 함수부터 함미까지 길이인 전장은 30~300m까지이다. 어선의 선체색깔은 일반적으로 흰색과 청색이며, 전장은 1~50m이다. 여객선의 선체색깔은 흰색이 대부분이며, 전장은 50~300m이다. 상선의 선체 색깔은 적색, 파란색, 갈색 등 다양하며, 전장은 50m 이상이다. 선체 색깔은 SVM에서 군함을 식별하는데 사용된다.

[표 2-3] 선박 종류별 특징

구 분	군 함	어 선	여객선	상선
선체 색깔	회색	흰색, 청색	흰색	다 양
전 장	30~300m	1~50m	50~300m	50m 이상
조타실 위 치	함수로부터 1/3 지점	선 미	선 수	선수 또는 선미
사 진				

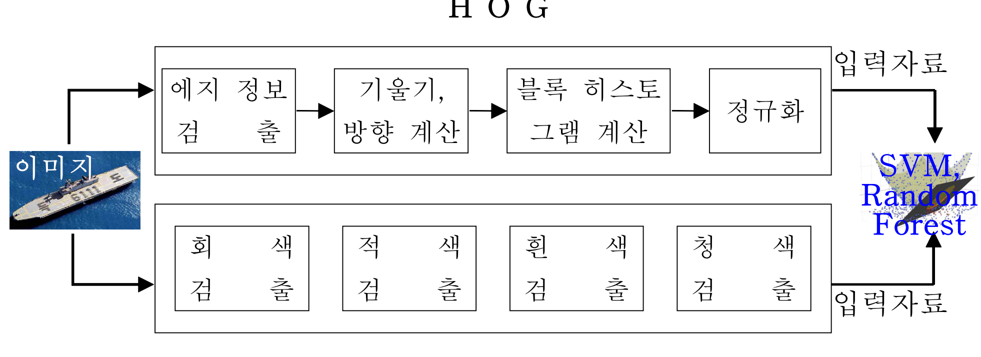
3. 이미지의 HOG 및 색상 정보

본 논문의 설험과정은 [그림 3-1]과 같다. 동영상에서 화면캡처를 하여 이미지를 저장한 후 실험을 해야 하나 자료를 획득하는 것이 제한되어 웹에서 이미지를 다운로드 하였다. 다운로드된 이미지의 HOG와 색상 정보는 SVM의 입력된 자료로 사용된다.



[그림 3-1] 실험과정

본 논문의 알고리즘은 [그림 3-2]와 같다.



[그림 3-2] 알고리즘

3.1. HOG Feature

HOG Feature는 Cell 히스토그램을 계산하고 정규화하는 것으로 나타난다. 본 논문에서는 HOG를 위하여 이미지를 252×156 pixel로 변환하여 실험을 하였다. HOG Cell 크기에 따라 SVM의 실험결과 값이 달라질 수 있다. 최초 HOG Cell 크기를 2×2, 3×3, 4×4로 설정하였으나 2×2로 실험을 할 때 이미지의 개수가 증가함에 따라 컴퓨터 RAM 용량이 작아 실험을 할 수 없어 HOG Cell 크기를 3×3, 4×4, 6×6 세 가지로 분류하여 실험을 하였다.

HOG Feature의 입력 값은 $[g][\theta][Cell\,size]$이며, HOG Feature의 결과 값은 $[w_i][h_j][b_z]$ 3개의 Vector로 표시되며, gradient 값과 방향 값, HOG Cell 크기에 따라 달라진다.

g : gradient 값

θ : 방향 값

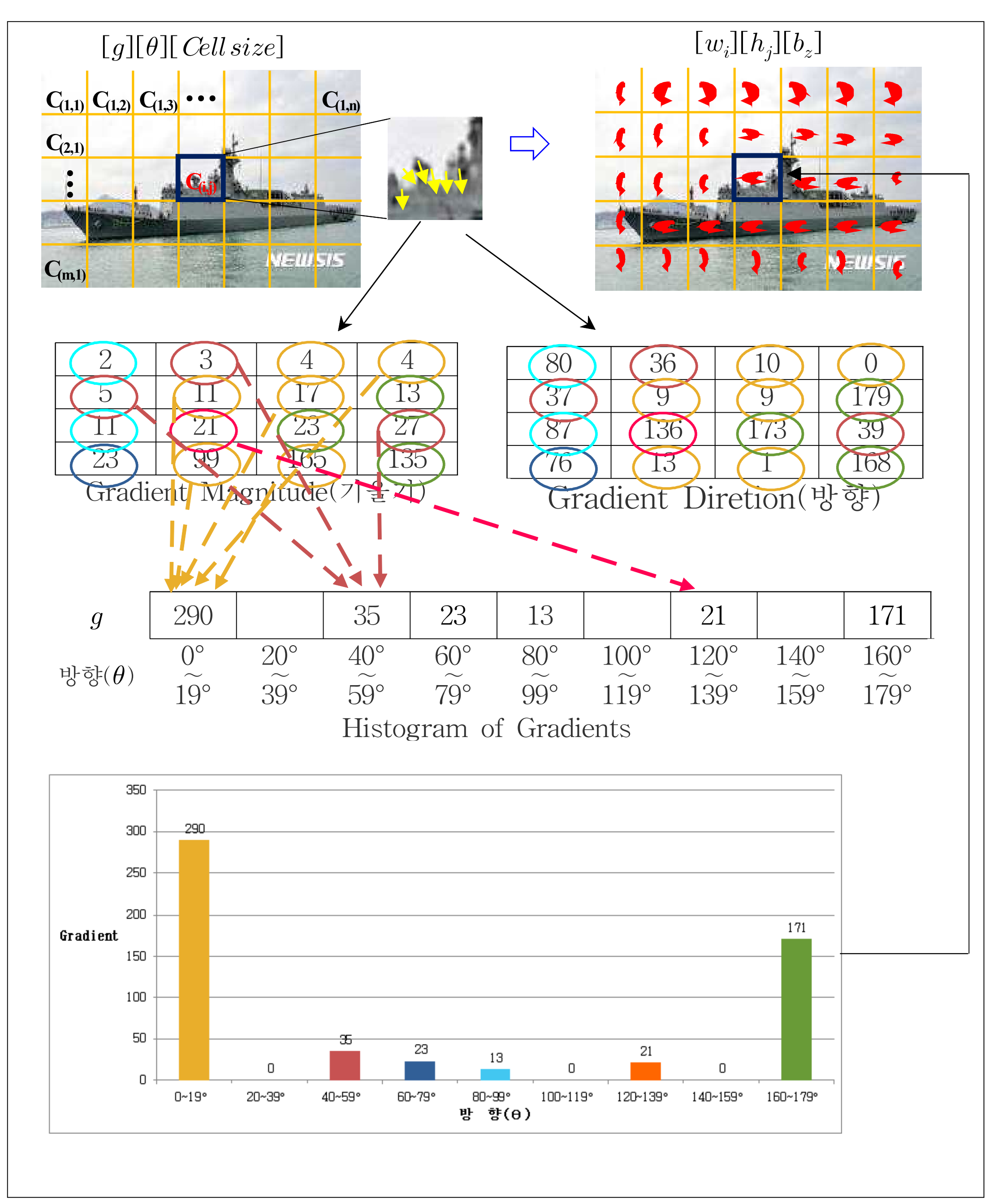
$Cell\,size$: HOG Cell 크기

w_i : 이미지의 폭에서 Cell 순서

h_i : 이미지의 높이에서 Cell 순서

b_z : 방향 0∼180°을 9등분(0°∼19°, 20°∼39°, 40°∼59°,…160°∼179°)

HOG Feature의 결과 값은 $[w_i][h_j][b_z]$의 구하는 과정은 [그림 3-3]과 같다.



[그림 3-3] HOG Feature 과정

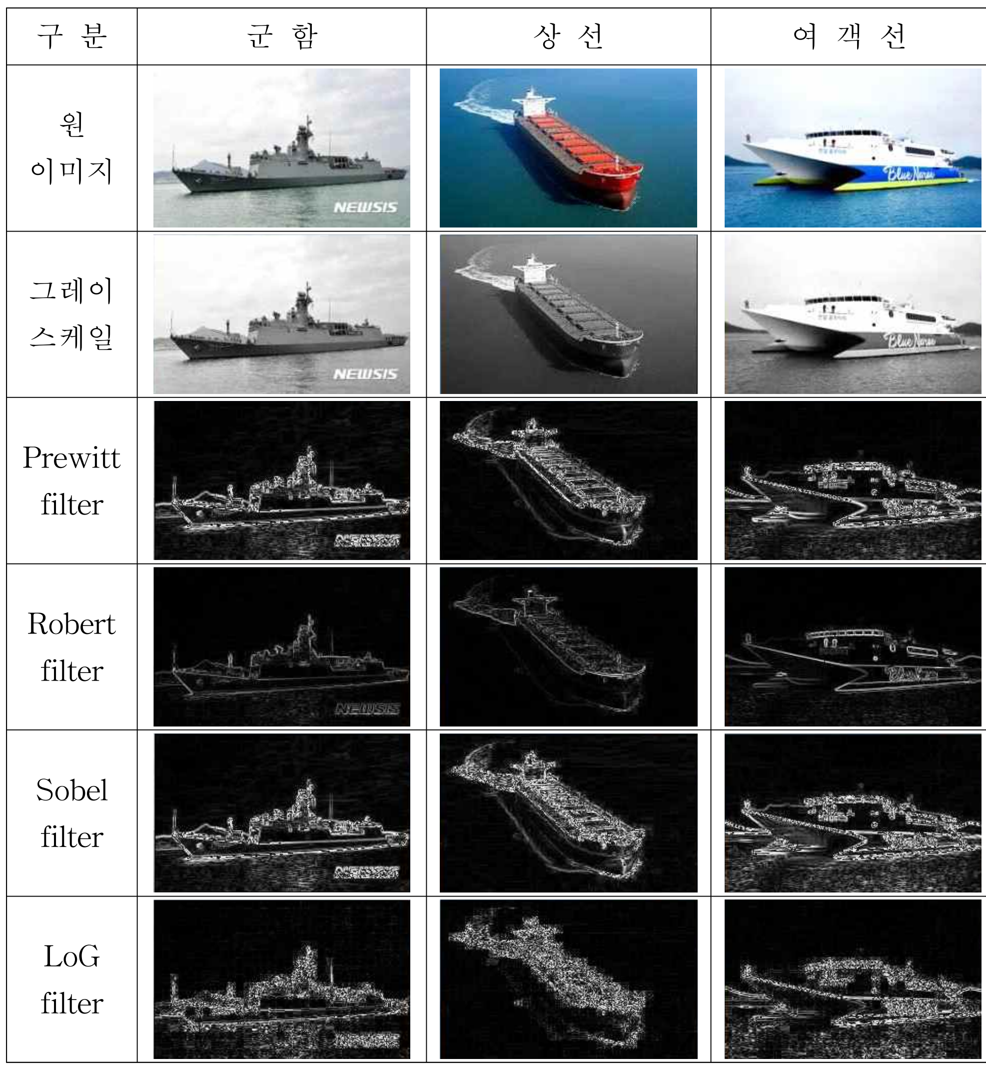
i, j 값은 HOG Cell 값에 따라 달라지고 z 값은 총 9개이다. Cell이 3×3, 4×4, 6×6 일 때 총 Cell 수는 [표 3-1]과 같다.

[표 3-1] HOG Cell 크기에 따른 Cell 수

구 분	i	j	총 CELL 수
3 × 3	252/3 = 84	156/3 = 52	84×52 = 4368
4 × 4	252/4 = 63	156/4 = 39	63×39 = 2457
6 × 6	252/6 = 42	156/6 = 26	84×52 = 1092

3.2. HOG를 위한 에지 정보 검출

HOG를 위해서 먼저 이미지를 그레이스케일로 변환을 해야한다. 이미지를 그레이스케일로 변환할 때 RGB의 평균값을 이용하였다. 이미지를 그레이스케일로 변환한 후 각 에지 정보 분포를 위해 4개의 filter 값을 적용하였으며, [그림 3-4]와 같다.



구 분	군 함	상 선	여 객 선
원 이미지			
그레이 스케일			
Prewitt filter			
Robert filter			
Sobel filter			
LoG filter			

[그림 3-4] Filter에 따른 에지 분포 이미지

3.3. 이미지 내 색상 검출

앞에서 설명하였듯이 군함의 선체 색깔은 95%이상이 회색이고, 기타 선박은 흰색, 청색, 적색 등이 주를 이룬다. 그래서 이미지 내에 회색의 양에 따라 군함과 기타 선박으로 구분할 수 있을 것이다. 마찬가지로 이미지 내에 흰색, 청색, 적색 등이 많으면 군함이 아닐 경우가 높을 것이다.

3.3.1. 회색 및 흰색 검출

회색은 RGB에서 R, G, B 값이 동일하다. 그러나 이미지 내에서 사람의 눈에 보이는 회색이 모두 R=G=B라 할 수 없을 것이다. 즉 R, G, B 값이 일부 다르더라도 이미지 내에 회색이 있다는 것이다. 이미지에서 회색을 검출하기 위해 R, G, B의 값을 설정해야 하는데 본 논문에서는 R>220, G>220, B>220 일때는 검정색, R<30, G<30, B<30 일때는 흰색으로 간주했다. 그리고 회색은 |R-G|<20, |G-B|<20, |B-R|<20으로 설정했다. 이미지에서 회색 값이 얼마나 있는가를 검출하기 위해서 각 이미지의 각 열의 회색 픽셀 개수를 구했다.

이미지에서 흰색 픽셀 값을 구하기 위한 슈도 코드는 [그림 3-5]와 같다.

```
int width     // 이미지의 폭 픽셀 크기
int heigt     // 이미지의 높이 픽셀 크기
for(i = 0; i<width; i++)
        #_white_pixel = 0     // 각 열을 시작할 때 최초 흰색 값
        total_#_white_pixel = 0     // 각 열을 시작할 때 최초 흰색 총합
    for(j = 0; j<height; j++)
         if R<30 and G<30 and B<30    // 픽셀이 흰색일 때
               #_white_pixel = #_gray_pixel + 1  //  각 열의 흰색 값 증가
               total_#_white_pixel = #_gray_pixel
```

[그림 3-5] 이미지의 흰색 값을 구하기 위한 슈도 코드

[그림 3-6]은 이미지에서 회색 픽셀 값을 구하기 위한 슈도 코드이다.

```
int width      // 이미지의 폭 pixel 크기
int heigt      // 이미지의 높이 pixel 크기
for(i = 0; i<width; i++)
        #_gray_pixel = 0      // 각 열을 시작할 때 최초 회색 값
        total_#_gray_pixel = 0      // 각 열을 시작할 때 최초 회색 총합
   for(j = 0; j<height; j++)
         if R>220 and G>220 and B>220  // 픽셀이 검정색일 때
              total_#_gray_pixel = #_gray_pixel
                                        //  각 열의 회색 값은 증가하지 않는다
         if R<30 and G<30 and B<30   // 픽셀이 흰색일 때
              total_#_gray_pixel = #_gray_pixel
                                        //  각 열의 회색 값은 증가하지 않는다
         if |R-G|<20 and |G-B|<20 and |B-R|<20  // 픽셀이 회색일 때
              #_gray_pixel = #_gray_pixel + 1      //  각 열의 회색 값 증가
              total_#_gray_pixel = #_gray_pixel   // 각 열의 회색 픽셀 총합
```

[그림 3-6] 이미지의 흰색 값을 구하기 위한 슈도 코드

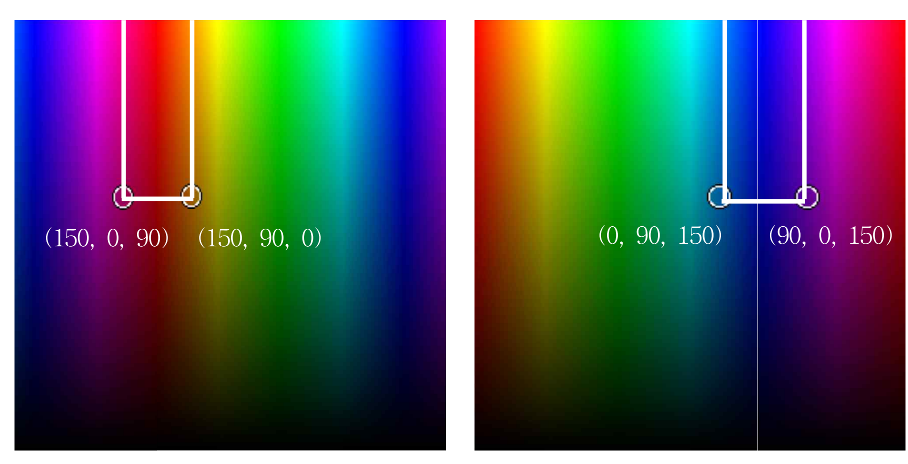
[그림 3-7]는 이미지에서 회색과 흰색 값을 검출한 것이다.

구 분	군 함	상 선	여 객 선
원 이미지			
회 색			
흰 색			

[그림 3-7] 이미지 내 회색 및 흰색 검출

3.3.2. 적색 및 청색 검출

적색 검출 및 청색 검출의 방식은 유사하다. 적색은 RGB에서 (255, 0, 0)이다. 이미지에서 적색을 검출하기 위해 R>150, G<90, B<90으로 범위를 설정하였다. 적색 색상은 [그림 3-8]과 같다. 청색은 RGB에서 (0, 0, 255)이다. 적색검출과 비슷한 방법으로 이미지에서 청색을 검출하기 위해 R<90, G<90, B>150으로 범위를 설정하여였으며, [그림 3-9]와 같다.



[그림 3-8] 적색의 색상 범위　　[그림 3-9] 청색의 색상 범위

[그림 3-10]은 이미지에서 적색 픽셀 값을 구하기 위한 슈도 코드이다.

```
int width     // 이미지의 폭 픽셀 크기
int heigt     // 이미지의 높이 픽셀 크기
for(i = 0; i<width; i++)
   for(j = 0; j<height; j++)
        if R>1500 and G<90 and B<90          // 픽셀이 적색일 때
             #_red_pixel = #_red_pixel + 1     //  각 열의 적색 값 증가
             total_#_red_pixel = #_red_pixel   // 각 열의 적색 pixel 총합
```

[그림 3-10] 이미지의 적색 값을 구하기 위한 슈도 코드

[그림 3-11]은 이미지에서 청색 픽셀 값을 구하기 위한 슈도 코드이다.

```
int width      // 이미지의 폭 픽셀 크기
int heigt      // 이미지의 높이 픽셀 크기
for(i = 0; i<width; i++)
   for(j = 0; j<height; j++)
        if R<90 and G<90 and B>150         // 픽셀이 청색일 때
             #_blue_pixel = #_blue_pixel + 1     //  각 열의 청색 값 증가
             total_#_blue_pixel = #_blue_pixel    // 각 열의 청색 픽셀 총합
```

[그림 3-11] 이미지의 청색 값을 구하기 위한 슈도 코드

[그림 3-12]는 이미지에서 적색과 청색 값을 검출한 것이다.

구 분	군 함	상 선	여 객 선
원 이미지			
적 색	없 음		없 음
청 색	없 음	없 음	

[그림 3-12] 이미지 내 적색 및 청색 검출

4. 실험 및 평가

4장 '실험 및 평가'에서는 이미지의 HOG 및 회색, 흰색, 적색, 청색 등을 SVM의 입력자료로 사용한다. 앞에서 설명하였듯이 이미지의 에지 분포 정보를 알아내기 위해 Sobel filter 등 4개의 filter 값이 사용된다. 분석도구는 WEKA(Waikato Environment for Knowledge Analysis) 3.6.9를 사용한다. WEKA 버전은 3.8.1 버전까지 나왔지만 SVM은 3.6.9 버전까지만 지원이 되고 있다. 그리고 SVM의 성능을 비교하기 위하여 Naïve Bayes Classification(나이브 베이즈 분류), J48, RandomForest 알고리즘도 같이 실험하였다. 본 실험은 JAVA 프로그래밍 언어로 수행되었다.

4.1. 실험 데이터

본 논문은 동영상 내 군함을 검출하는 것이다. 그러면 실질적으로 바다에서 항해하고 있는 선박들을 촬영하여 군함인지 아닌지를 식별해야 한다. 그러나 육상에서 해상으로 나가 선박들을 촬영하기에는 시간이 많이 걸리고 다수의 자료를 확보하기에 어려움이 있다. 그래서 동영상에서 화면 캡처하여 이미지를 저장하는 것을 웹에서 이미지를 다운로드하는 것으로 대신하였다. 웹에서 이미지를 다운로드하면 다수의 다른 이미지를 확보할 수 있는 장점이 있는 반면 해상도가 낮거나 이미지 내에 글자가 포함되는 등의 단점도 있었다.

군함은 [그림 4-1]과 같이 군함의 종류별로 항공모함 등 총 1,200장 다운로드 하였다.

[그림 4-1] 군 함

기타 선박은 [그림 4-2]와 같이 상선, 여객선, 어선 등으로 총 1,200장 다운로드 하였다.

[그림 4-2] 상선, 여객선, 어선

4.2. 실험환경

본 실험에서는 앞에서 설명하였듯이 웹에서 다운로드한 이미지를 사용한다. 다운로드한 이미지는 252×156 크기로 인코딩 한 후 실험에 적용하였다. 실험환경은 CPU Intel(R) 64bit core(TM) 2 Quad CPU 2.5GHz RAM 8G를 사용하였다. 실험은 웹에서 다운로드한 4800개의 이미지를 사용하였다. 군함 2,400개와 기타 선박 2,400개이다. 이미지를 군함 3,000개, 기타 선박 3,000개 등 총 6,000개 이상을 사용했을 때는 컴퓨터 RAM 용량이 작아 실험을 더 진행하지 못한 것을 아쉽게 생각한다.

4.3. 평가지표

군함을 검출하는 데 있어서 제일 중요한 것은 정확도이다. 즉 참인 것을 참이라고 식별하고 거짓인 것을 거짓으로 식별해야 한다. 군에서 경계임무를 수행할 때 거짓을 참이라고 한 경우에는 거짓을 거짓으로 식별할 수 있지만 참을 거짓으로 식별하여 아무 조치도 하지 않는다면 문제가 발생할 수 있다. 적을 적이라고 식별했을 때는 최상이다. 적이 아닌 것을 적이라고 했을 때는 다시 적이 아니라고 판단하면 된다. 그러나 적을 적이 아니라고 했을 때는 우리에게 위협이 될 것이다.

본 논문에서는 성능평가를 할 때 [표 4-1] 같이 혼동행렬을 사용할 것이다.

$$\text{정해률(Accuracy)} = \frac{TP+TN}{TP+FP+TN+FN} \tag{4-1}$$

[표 4-1] 혼동행렬(Confusion Matrix)

구 분		예 측	
		긍 정	부 정
실 제	참	True Positive (TP)	False Negative (FN)
	거짓	False Positive (FP)	True Negative (TN)

참 긍정률(recall rate, TPR, True Positive Rate) = $\frac{TP}{TP+FN}$ (4-2)

거짓 음성률(FNR, False Negative Rate) = $\frac{FN}{TP+FN}$ (4-3)

정밀도(Precision) = $\frac{TP}{TP+FP}$ (4-4)

거짓 긍정률(FPR, False Positive Rate) = $\frac{FP}{FP+TN}$ (4-5)

참 음성률(TNR, True Negative Rate) = $\frac{TN}{TN+FP}$ (4-6)

에러율(Error rate) = $\frac{FP}{TP+FP+TN+FN}$ (4-7)

여기서 TP 는 '참 긍정'으로 실제로 군함인데 알고리즘에서도 군함으로 예측한 경우, FN 은 '거짓 부정'으로 실제로 군함인데 군함이 아닌 것으로 예측한 경우, FP 는 '거짓 긍정'으로 실제로는 군함이 아닌데 군함으로 예측한 경우, TN 은 '참 부정'으로 실제로는 군함이 아닌데 군함이 아닌 것으로 예측한 경우를 의미한다.

본 논문에서는 식 4-1인 정해률($(TP+TN)/(TP+FP+TN+FN)$), 식

4-2 참 긍정율($TP/(TP+FN)$), 식 4-4 정밀도($TP/(TP+FP)$)를 알아본다. 정해률은 군함인 것을 군함으로 식별하고, 군함이 아닌 것을 군함이 아닌 것으로 식별한 것이다. 참 긍정률은 군함 중에서 군함이라고 식별한 것이며, 정밀도는 군함이라고 했는데 실제로 군함인 것을 말한다.

군의 경계임무에 있어서는 정해률도 높아야 하지만 군함인 것 중에 실제로 군함으로 식별한 것, 즉 참 긍정율($TP/TP+FN$)이 더 중요하다. 왜냐하면 앞에서도 설명했듯이 군에서는 적을 적으로 식별하는 것이 경계임무에 있어 중요하기 때문이다. 또한 정밀도($TP/(TP+FP)$)도 중요할 것이다. 왜냐하면 군함이라고 했는데 군함이 아니면 경계임무를 수행함에 있어 피로도가 높을 것이기 때문이다.

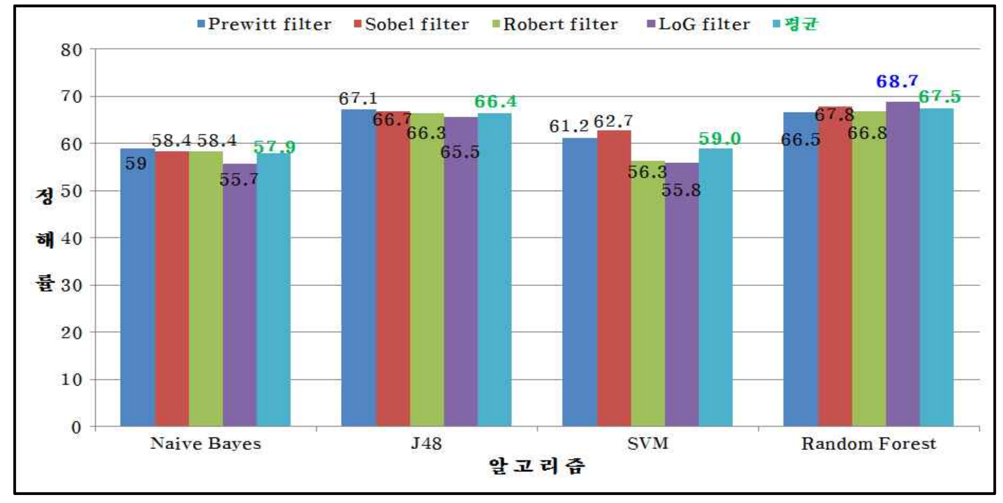
4.4. 실험결과

4.4.1. 정해율(Accuracy)

본 논문에서는 정해률을 두 가지 방면으로 실험하여 결과를 도출하였다. 첫 번째는 HOG만을 이용했으며 두 번째는 HOG와 이미지 색상 값으로 실험을 하였다.

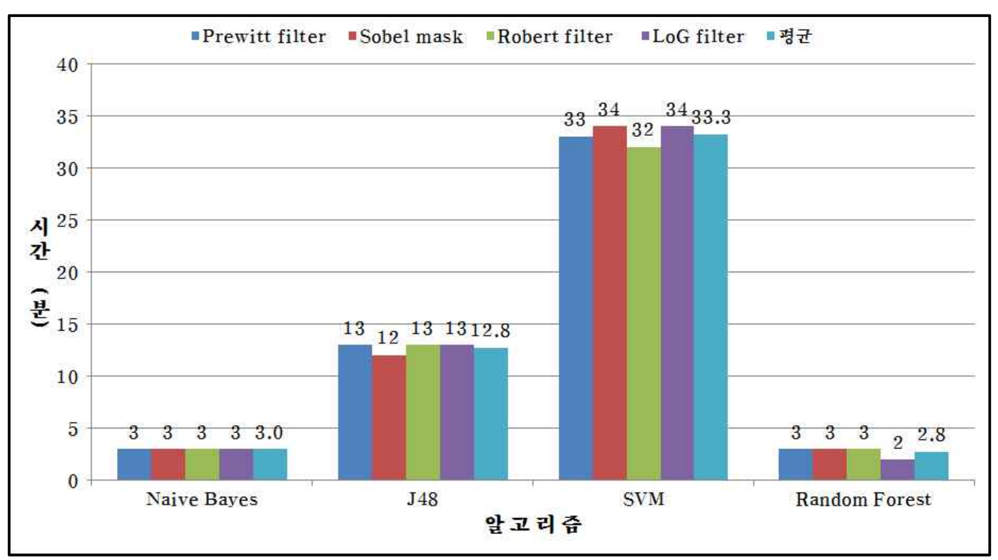
4.4.1.1. HOG만을 이용한 알고리즘 평가

HOG의 Cell은 4×4, BIN은 9를 사용였고 이미지는 총 2,400개, 즉 군함 이미지 1,200개, 기타 선박 이미지 1,200개를 이용하여 실험을 하였다. 실험 결과 정해률이 55.7~68.7%로 전반적으로 높지 않았다. Random Forest가 평균 67.5%로 가장 높았고 J48이 66.4%, SVM은 59%, Naive Bayes는 57.9% 순으로 결과가 나왔으며 [그림 4-3]과 같다.



[그림 4-3] HOG만을 이용한 filter별 알고리즘 정해률

결과를 도출할 때 소요되는 시간도 같이 측정을 하였는데 Naive Bayes와 Random Forest가 3분으로 가장 짧았고 J48은 평균 12분 50초, SVM은 33분 20초로 가장 길었으며 [그림 4-4]와 같다.



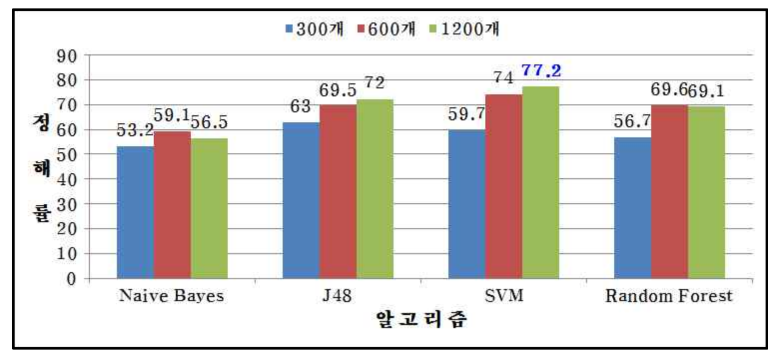
[그림 4-4] HOG만을 이용한 filter별 알고리즘 소요 시간

HOG만을 이용했을 때는 정해률과 소요시간을 고려했을 때 Random Forest가 가장 성능이 우수하였다. SVM은 정해률은 세 번째였으나 시간은 가장 많이 소요되었다.

4.4.1.2. HOG와 색상을 이용한 알고리즘 평가

HOG와 색상을 이용한 실험은 다양한 방법으로 실시하였다. 먼저 HOG의 Cell은 3×3, 4×4, 6×6 3가지를 사용했으며, HOG의 BIN은 9를 사용하였다. HOG의 2×2 Cell은 이미지의 개수가 300개, 600개 일 때는 실험이 가능하였으나 1,200개일 때는 컴퓨터 RAM 용량 제한으로 실험을 하지 못해 실험 대상에서 제외시켰다. 또한 이미지 개수를 300개, 600개, 1,200개로 구분하여 실시하였으며, 이미지의 에지 분포 정보를 알아내는 4개의 filter, Prewitt filter, Robert filter, Sobel filter, LoG filter를 각각 사용하였다.

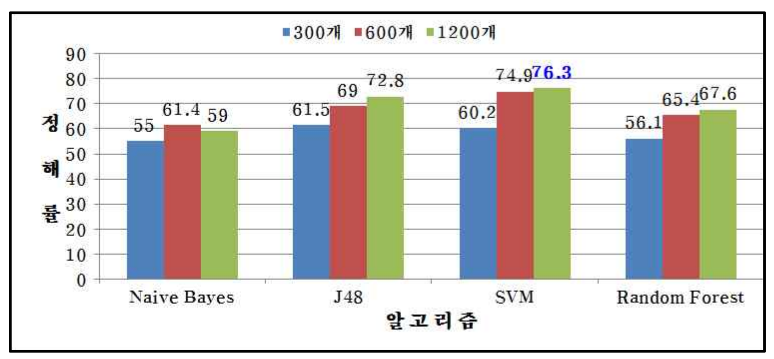
먼저 HOG Cell 3×3과 Prewitt filter를 이용한 실험 결과는 [그림 4-5]와 같다. 이미지 개수가 증가하면서 4개의 알고리즘의 정해율이 높아졌다. 하지만 Naive Bayes와 Random Forest 알고리즘은 1200개에서 정해률이 다소 낮아졌다. 정해율은 SVM 알고리즘에서 이미지 1,200개일 때 77.2%로 가장 높았다.



[그림 4-5] HOG Cell 3×3, Prewitt filter

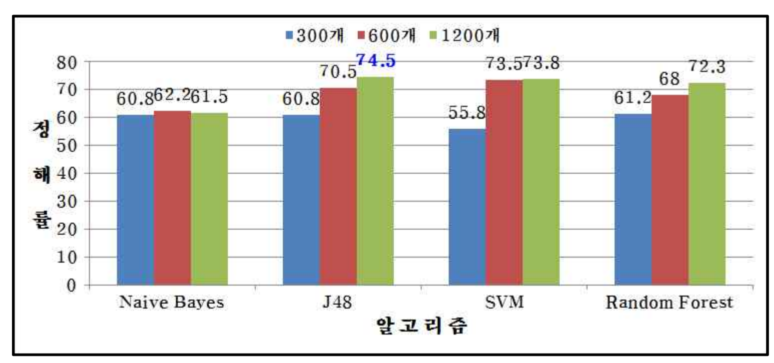
HOG Cell 4×4와 Prewitt filter를 이용한 실험 결과는 [그림 4-6]과 같다. HOG Cell 3×3일 때와 마찬가지로 이미지의 개수가 늘어남에 따라

정해률이 높아졌다. SVM 알고리즘에서 이미지 1,200개 일 때 76.3%로 정해률이 가장 높았으나 HOG Cell 3×3일 때 보다는 정해률이 0.9% 낮았다.



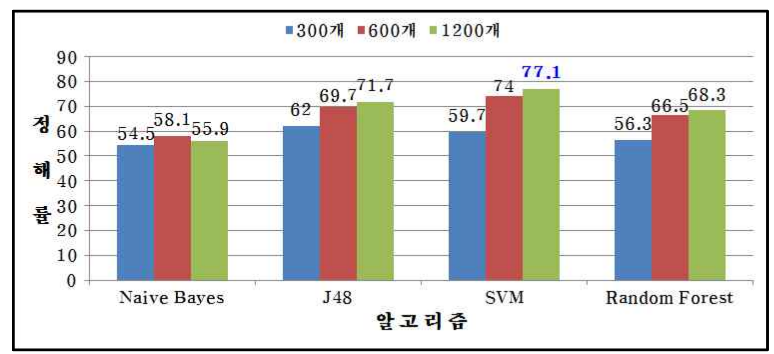
[그림 4-6] HOG Cell 3×3, Prewitt filter

HOG Cell 6×6과 Prewitt filter를 이용한 실험 결과는 [그림 4-7]과 같다. 여기서도 이미지의 개수가 늘어남에 따라 정해률이 높아졌지만, J48과 SVM의 정해률은 HOG Cell 3×3, 4×4보다는 결과가 상대적으로 낮았으나, Naive Bayes와 Random Forest의 정해율은 HOG Cell 3×3, 4×4의 정해률보다 더 높게 나왔다. J48 알고리즘에서 이미지 1,200 일 때 정해률이 74.5%로 가장 높았다.



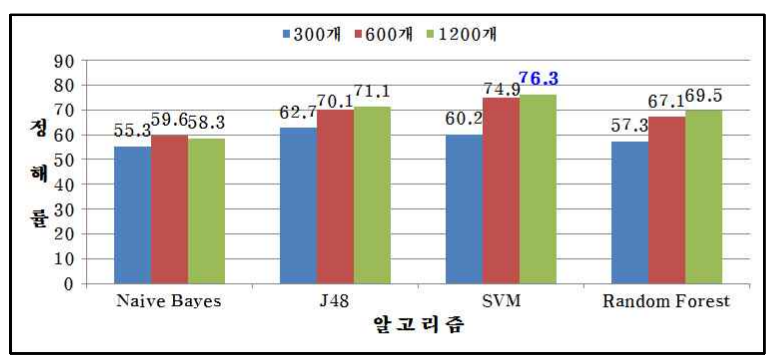
[그림 4-7] HOG Cell 6×6, Prewitt filter

다음은 Robert filter 실험 결과이다. HOG Cell 3×3, Robert filter 실험결과는 [그림 4-8]과 같고, Prewitt filter와 마찬가지로 이미지의 개수가 증가함에 따라 정해률이 높아졌다. SVM 알고리즘에서 이미지 1,200일 때 77.1%로 가장 높았고, Prewitt filter보다는 정해률이 0.1% 낮았다.



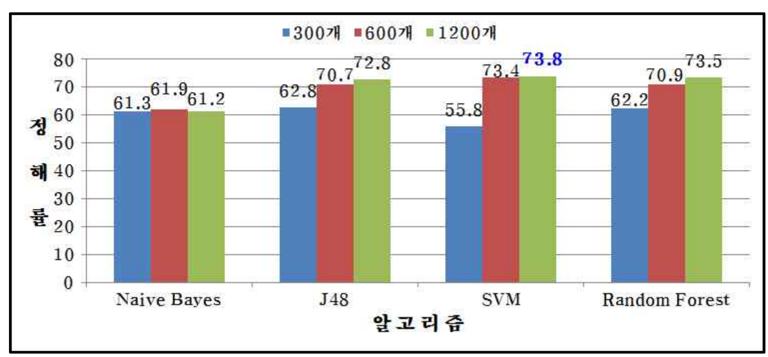
[그림 4-8] HOG Cell 3×3, Robert filter

HOG Cell 4×4, Robert filter를 이용한 실험 결과는 [그림 4-9]와 같다. SVM 알고리즘에서 이미지 1,200개 일 때 정해률이 76.3%로 가장 높았다.



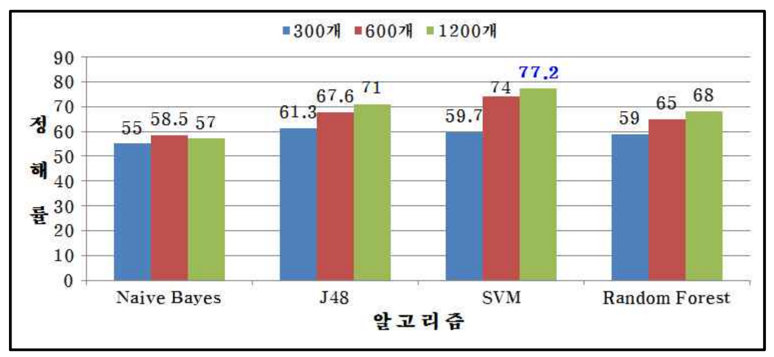
[그림 4-9] HOG Cell 4×4, Robert filter

HOG Cell 6×6, Robert filter를 이용한 실험 결과는 [그림 4-10]과 같다. Prewitt filter와 마찬가지로 J48과 SVM은 HOG Cell 3×3, 4×4보다는 결과가 상대적으로 낮았으나, Naive Bayes와 Random Forest는 높게 나왔다. SVM 알고리즘에서 이미지 1,200 일 때 정해률이 73.8%로 가장 높았다.



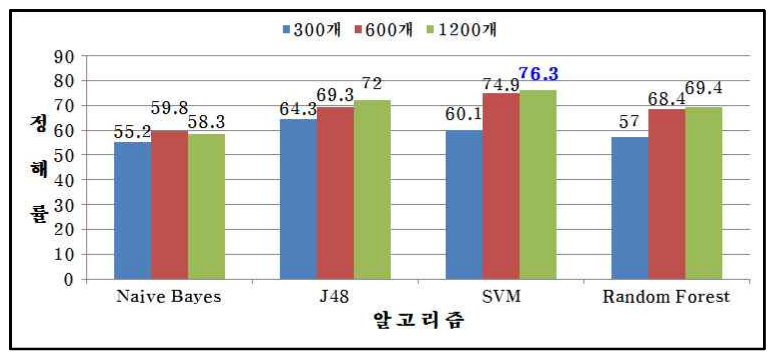
[그림 4-10] HOG Cell 6×6, Robert filter

세 번째로 Sobel filter 실험 결과이다. HOG Cell 3×3, Sobel filter 실험결과는 [그림 4-11]과 같고, 정해율은 SVM 알고리즘에서 이미지 1,200개 일 때 77.2%로 가장 높았다.



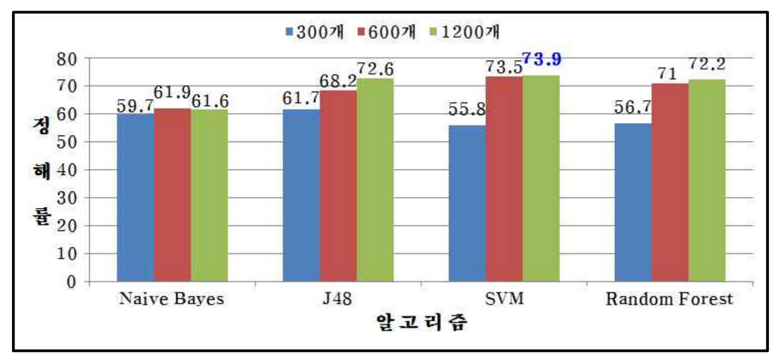
[그림 4-11] HOG Cell 3×3, Sobel filter

HOG Cell 4×4, Sobel filter 실험결과는 [그림 4-12]와 같고, 정해율은 SVM 알고리즘에서 이미지 1,200개 일 때 76.3%로 가장 높았다.



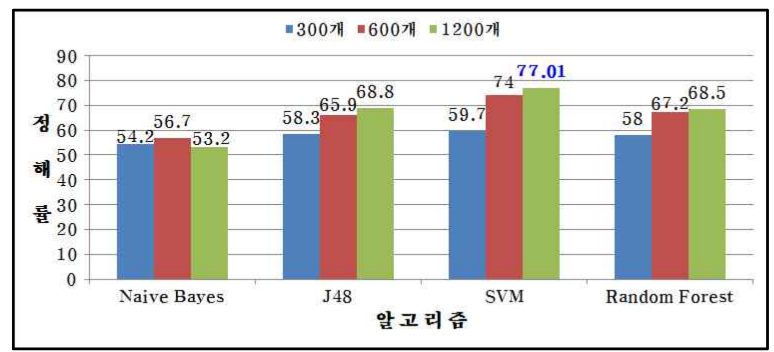
[그림 4-12] HOG Cell 4×4, Sobel filter

HOG Cell 6×6, Sobel filter 실험결과는 [그림 4-13]과 같고, 정해율은 SVM 알고리즘에서 이미지 1,200 일 때 73.9%로 가장 높았다. SVM 알리즘은 Prewitt filter, Robert filter와 마찬가지로 HOG Cell 3×3, 4×4 실험결과보다 낮게 나왔고, Naive Bayes 알고리즘은 HOG Cell 3×3, 4×4 보다 높게 나왔다. 그러나 J48, Random Forest는 Prewitt filter, Robert filter와 다르게 HOG Cell 3×3, 4×4보다 정해률이 높게 나왔다.



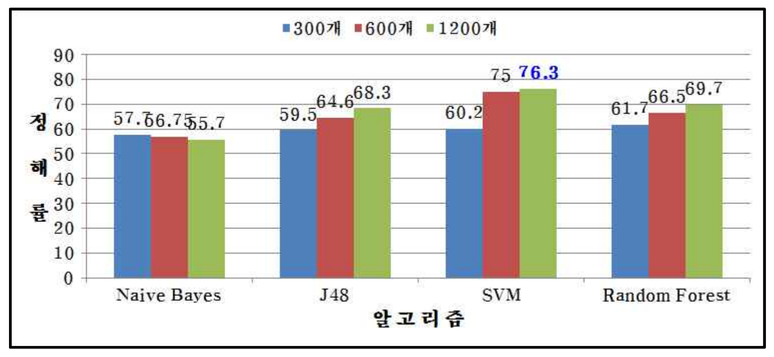
[그림 4-13] HOG Cell 6×6, Sobel filter

마지막으로 LoG filter 실험 결과이다. HOG Cell 3×3, LoG filter 실험결과는 [그림 4-14]와 같고, 정해율은 SVM 알고리즘, 이미지 1,200개에서 77.01%로 가장 높았다. LoG filter의 정해율이 Prewitt filter, Robert filter, Sobel filter보다 근소하지만 제일 낮은 결과가 나왔다.



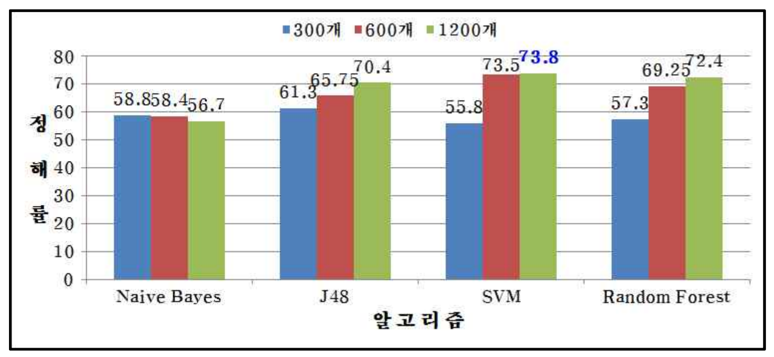
[그림 4-14] HOG Cell 3×3, LoG filter

HOG Cell 4×4, LoG filter 실험결과는 [그림 4-15]와 같고, 정해율은 SVM 알고리즘에서 이미지 1,200개 일 때 76.3%로 가장 높았다.



[그림 4-15] HOG Cell 4×4, LoG filter

HOG Cell 6×6, LoG filter 실험결과는 [그림 4-16]과 같고, 정해률은 SVM 알고리즘에서 이미지 1,200개 일 때 73.8%로 가장 높았다.



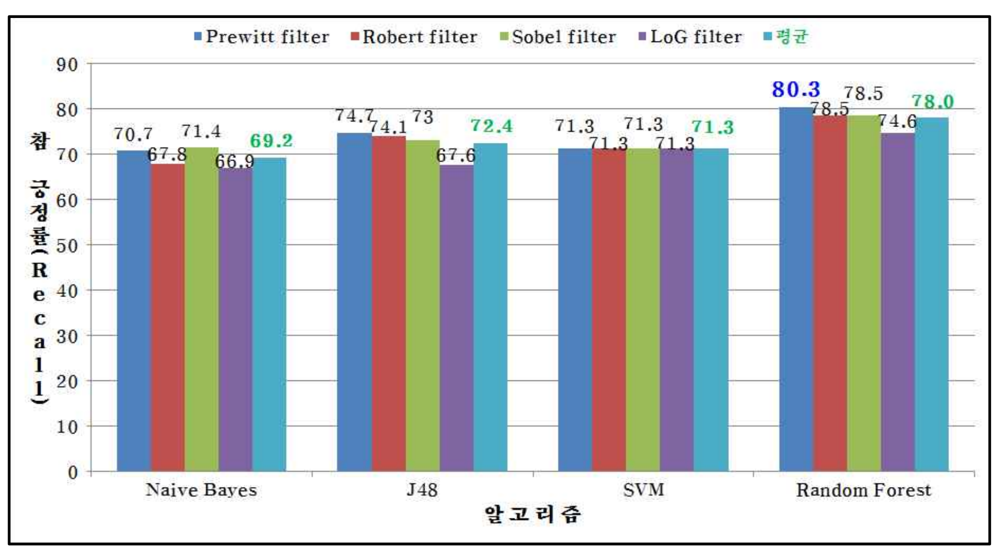
[그림 4-16] HOG Cell 6×6, LoG filter

종합해 보면, 이미지의 개수가 늘어남에 따라 SVM 알고리즘의 결과가 가장 좋았으며, 정해률은 최대 77.2%였다. 그러나 상대적으로 시간이 많이 걸리는 단점이 있었으며, 특히 이미지의 개수가 증가할 때 훨씬 더 많이 소요되었다.

4.4.2. 참 긍정율(Recall rate)

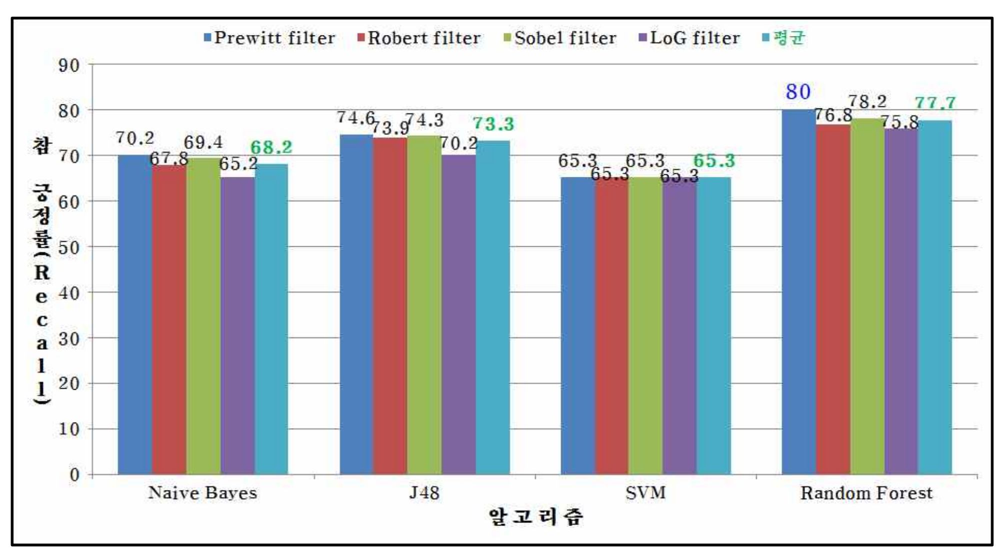
참 긍정율($TP/(TP+FN)$)은 군함인 것을 군함으로 식별해 낸 것이다. 군에서 경계임무에 있어 가장 중요한 부분이다. 참 긍정율의 실험은 이미지 1,200개, Prewitt, Robert, Sobel, LoG filter, HOG Cell 3×3, 4×4, 6×6을 사용하였다

[그림 4-17]은 HOG Cell 3×3에 대한 참 긍정율의 결과이다. 참 긍정율의 결과는 정해률과 다르게 Random Forest > J48 > SVM > Naive Bayes 알고리즘 순이다. 참 긍정율의 최고 값은 Random Forest 알고리즘, Prewitt filter에서 80.3%이다.



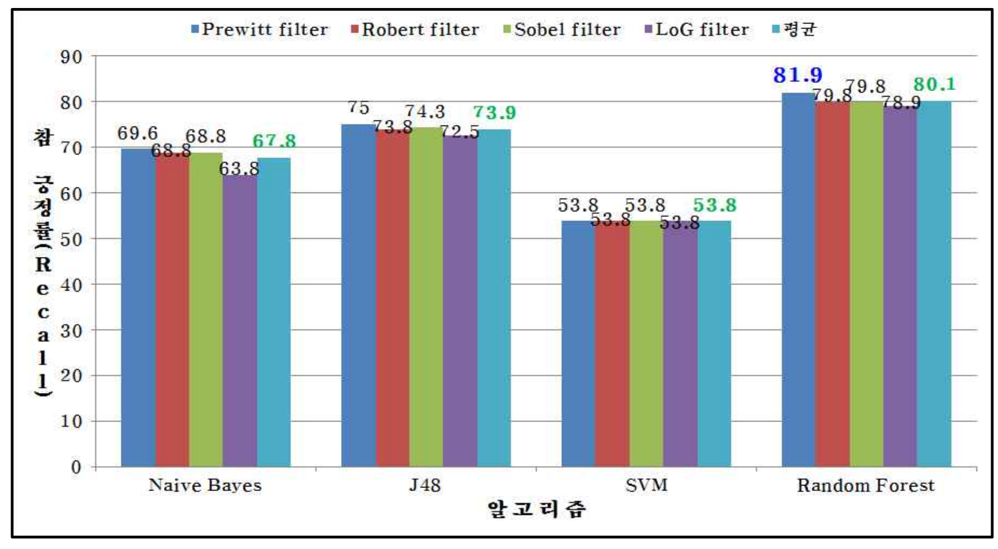
[그림 4-17] HOG Cell 3×3에 대한 참 긍정율

다음 페이지의 [그림 4-18]은 HOG Cell 4×4에 대한 참 긍정율 결과이다. HOG Cell 3×3 참 긍정율의 결과와 마찬가지로 Random Forest에서 평균 77.7%로 가장 좋았다. 그러나 SVM의 알고리즘은 정해률과 다르게 평균 65.3%로 가장 좋지 않은 결과 값이 나왔다. 가장 높은 값은 Random Forest, Prewitt filter에서 80%이다.



[그림 4-18] HOG Cell 4×4에 대한 참 긍정율

[그림 4-19]는 HOG Cell 6×6에 대한 참 긍정율 결과이다. HOG Cell 6×6 참 긍정율의 결과는 Random Forest > J48 > Naive Bayes > SVM 알고리즘 순이다. 가장 높은 값은 Random Forest, Prewitt filter에서 81.9%이고 가장 낮은 값은 SVM에서 53.8%이다.

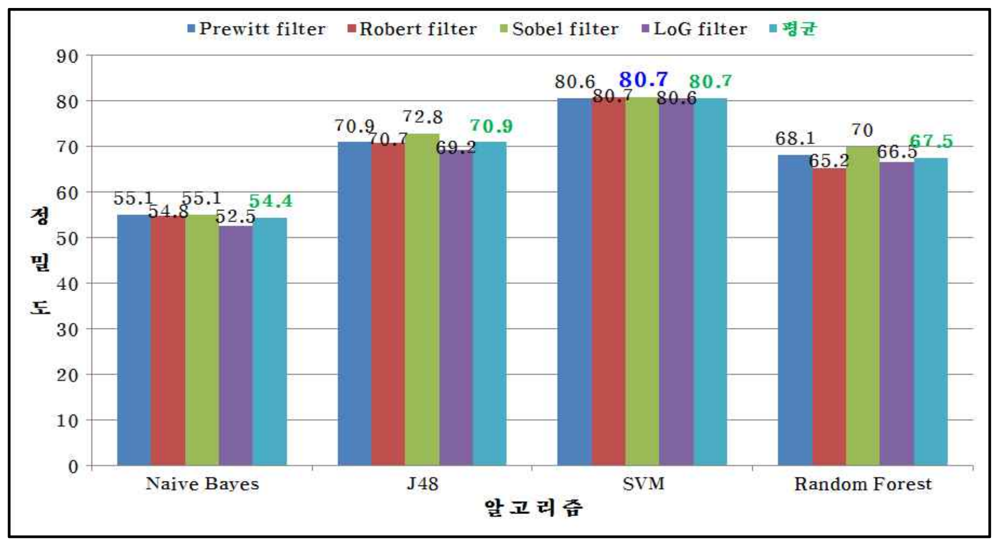


[그림 4-19] HOG Cell 6×6에 대한 참 긍정율

4.4.3. 정밀도(Precision)

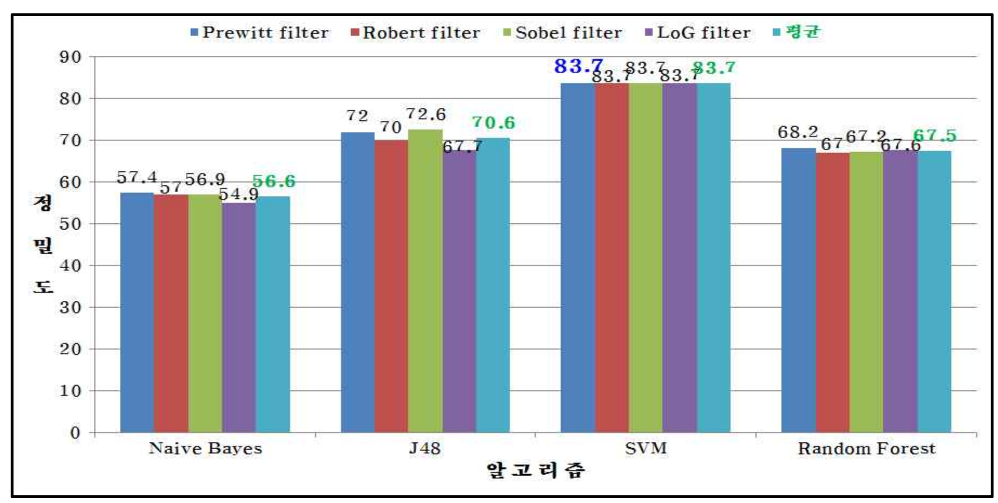
정밀도($TP/(TP+FP)$)는 군함이라고 했는데 실제로 군함인 것을 말한다. 앞에서도 설명했지만 군함이라고 했는데 군함이 아니면 알고리즘의 신뢰문제가 생기며, 경계임무자가 피곤함을 많이 느낄 것이다. 정밀도의 실험은 참 긍정률과 마찬가지로 이미지 1,200개, Prewitt, Robert, Sobel, LoG filter, HOG Cell 3×3, 4×4, 6×6을 사용하였다

[그림 4-20]은 HOG Cell 3×3에 대한 정밀도의 결과이다. 정밀도의 결과는 SVM > J48 > Random Forest > Naive Bayes 알고리즘 순이다. 정밀도의 최고 값은 SVM 알고리즘에서 80.7%이다. SVM은 모든 filter에서 80.7%로 filter와 무관하였다.



[그림 4-20] HOG Cell 3×3에 대한 정밀도

다음 페이지의 [그림 4-21]은 HOG Cell 4×4에 대한 정밀도의 결과이다. 정밀도의 결과는 HOG Cell 3×3에 대한 정밀도의 결과와 마찬가지로 SVM > J48 > Random Forest > Naive Bayes 알고리즘 순이다. 정밀도의 최고 값은 SVM 알고리즘에서 83.7%이다.



[그림 4-21] HOG Cell 4×4에 대한 정밀도

[그림 4-21]은 HOG Cell 6×6에 대한 정밀도의 결과이다. 정밀도의 결과는 HOG Cell 3×3, 4×4에 대한 정밀도의 결과와 마찬가지로 SVM > J48 > Random Forest > Naive Bayes 알고리즘 순이다. 정밀도의 최고값은 SVM 알고리즘에서 89.7%이다.

[그림 4-22] HOG Cell 6×6에 대한 정밀도

4.4.4. 종합

실험 결과를 종합해보면 정해률은 SVM 알고리즘, HOG Cell 3×3에서, 참 긍정율은 Random Forest 알고리즘, HOG Cell 6×6에서, 정밀도는 SVM 알고리즘, HOG Cell 6×6에서 결과 값이 가장 높았다. 정해률, 참 긍정률, 정밀도의 가장 큰 값은 [표 4-2]와 같다.

[표 4-2] 정해률, 참 긍정률, 정밀도의 가장 큰 값

구 분	SVM, HOG Cell 3×3	Random Forest, HOG Cell 6×6	SVM, HOG Cell 6×6
정해률	**77.2**	72.6	73.8
참 긍정율	71.3	**81.9**	53.8
정밀도	80.6	69.1	**89.7**

[표 4-3]은 정해률, 참 긍정률, 정밀도가 가장 클 때의 혼동행렬을 나타낸다. 경계에 임무에 있어서 군함을 군함이라고 식별한 것, 즉 TP가 가장 큰 값이 좋다. Random Forest, HOG Cell 6×6에서 참 긍정률이 81.9%로 이미지 개수 983개로 가장 높으나 [표 4-2], [표 4-3]과 같이 정밀도가 69.1%로 군함이 아닌 것을 군함이라고 한 것도 30.9%, 440개로 높았다. 그러나 SVM, HOG 3×3은 참 긍정률이 Random Forest 보다 10% 낮으나 정밀도는 11.5% 높다.

[표 4-3] 정해률, 참 긍정률, 정밀도의 가장 큰 값의 혼동행렬

SVM, HOG Cell 3×3		Random Forest, HOG Cell 6×6		SVM, HOG Cell 6×6	
TP : 856	FN : 334	TP : 983	FN : 217	TP : 646	FN : 554
FP : 204	TN : 996	FP : 440	TN : 760	FP : 74	TN : 1126

5. 결론

5.1. 요약 및 의의

본 논문에서는 Support Vector Machine을 이용한 동영상 내 군함을 검출하는 것을 제안하였다. 먼저 군함과 비슷한 에지 분포를 가지고 있는가를 식별하기 위한 HOG와 군함의 색상을 가지고 있는가 없는가를 RGB 색상표를 통해 이미지 내에 군함이 있는가 없는가를 식별하였다. 이미지의 에지 분포 정보를 알기 위해 Prewitt filter, Robert filter, Sobel filter, LoG filter를 사용했다. 또한 이미지에 군함의 선체 색깔인 회색과 기타 선박의 선체 색깔인 흰색, 적색, 청색이 얼마나 있는가를 검출하였다. 이미지의 에지 분포 정보와 색상 정보는 SVM의 입력자료로 사용되었으며 실험 결과 SVM 알고리즘에서 77.2%의 정해률을 나타났고, Random Forest 알고리즘에서 81.9%의 참 긍정률을 보였다.

실험은 정해률, 참 긍정률, 정밀도의 값을 산출하였다. HOG 입력 자료로 Cell 3×3, 4×4, 6×6 등 3가지로 구분하여 실험을 하였으며, 색상정보는 회색, 흰색, 적색, 청색 등을 사용하였다. Naive Bayes, J48, SVM, Random Forest 알고리즘으로 실험을 하였다.

정해률 산출은 두 가지로 진행되었다. 첫 번째는 HOG 자료만을 이용하였고, 두 번째는 HOG와 색상 정보를 함께 사용하였다. HOG만을 이용한 실험 결과는 Random Forest > J48 > SVM > Naive Bayes 알고리즘 순이었다. 그러나 최대 정해률이 68.7%로 비교적 낮게 나왔다. HOG와 색상 정보를 이용한 실험에서는 이미지의 개수가 300개일 때는

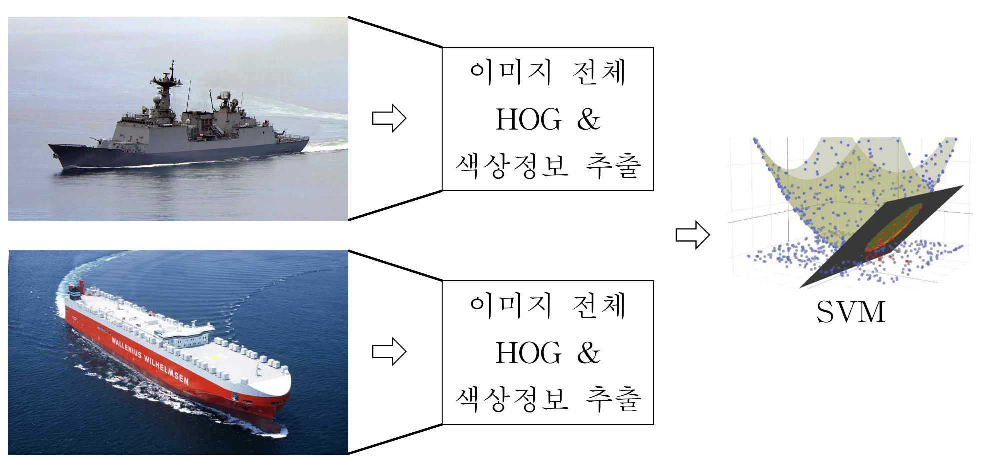
J48 알고리즘에서 다른 알고리즘보다 정해률 값이 근소하나마 높은 결과가 나왔으나 이미지의 개수가 600개, 1,200개로 증가할 때에는 SVM 알고리즘의 정해률 값이 상대적으로 높게 나왔다. HOG의 Cell 3×3, SVM 알고리즘에서 최대 77.2%의 결과가 나왔으며, 정밀도도 80.6%로 다른 알고리즘보다 상대적으로 높았다. 그러나 SVM의 알고리즘은 결과 값을 산출하는데 상대적으로 긴 시간이 소요되었다.

참 긍정율은 Random Forest에서 최대 81.9%의 값을 보였다. 그러나 정밀도는 69.1%로 낮게 나왔다.

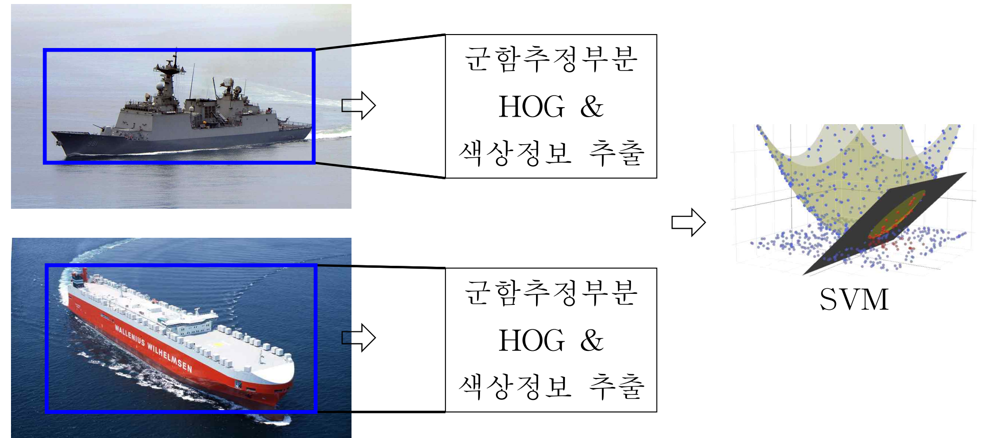
군대에서 경계는 매우 중요하다. 경계의 한 번의 실수가 작전의 승패를 좌지우지 하는 경우가 많이 있다. 군대에서는 경계를 함에 있어서 반드시 100%를 달성해야 한다. SVM의 참 긍정율 71.3%, Random Forest의 참 긍정율 81.9%는 100%에 비해 상당히 낮은 수치이다. 또한 경계임무에 있어서 71.3%, 81.9%는 용납될 수 없다. 하지만 군함 검출 시스템을 병행해서 사용한다면 경계임무에 도움이 될 것이라 생각한다.

5.2. 후속 과제

본 논문에서 군함 검출을 할 때 HOG와 색상정보를 검출함에 있어 이미지 전체를 사용하였다. 그러나 이미지 내에 군함과 비슷한 부분을 찾아내어 HOG와 색상정보를 알아낸다면 훨씬 결과 좋을 것이라 생각한다. 즉, 본 논문에서는 [그림 5-1]과 같이 실험을 하였으나 [그림 5-2]와 같이 실험을 하면 결과가 높게 나올 것이라 생각한다.



[그림 5-1] 본 논문에서의 실험방법



[그림 5-2] 제안하는 실험방법

2012년 스탠포드대학의 앤드류 응과 구글이 함께한 딥 러닝 프로젝트에서 16,000개의 컴퓨터 프로세서와 10억 개 이상의 Neural Networks 그리고 DNN(Deep Neural Networks)을 이용하여 유튜브에 업로드 되어있는 천만 개 넘는 비디오 중 고양이 인식에 성공하였다.[3] 군함 검출도 딥 러닝을 이용한다면 SVM이나 Random Forest보다 성능이 훨씬 좋을 수 있다. 또한 CNN(Convolutional Neural Network)은 다른 딥 러닝 구조들과 비교해서 영상, 음성 분야 모두에서 좋은 성능을 보여주기 때문에 군함 검출 결과도 좋을 것으로 판단된다.

3) 위키피디아, 딥 러닝, https://ko.wikipedia.org/wiki/%EB%94%A5_%EB%9F%AC%EB%8B%9D#cite_note-2 (2017. 6.20.)

참고 문헌

[1] 배재윤. (2015). 후보군 검출과 HOG 및 SVM을 이용한 선박 검출, 연세대학교, 16-29.

[2] Alasdair McAndrew. (2011). Introduction to digital image processing with matlab, Hanbit Media.

[3] Bishop. (2006). Christopher M. Pattern recognition and machine learning. springer.

[4] César Souza. (2010). Kernel Functions for Machine Learning Applications.

[5] Chang, Chih-Chung., Chih-Jen Lin. (2011). LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology (TIST) 2.3 : 27.

[6] Cortes, C., Vapnik, V. (1995). Support-Vector Networks. Machine Learning, 20, 273-297.

[7] D.M.J. Tax, R.P.W. Duin(2004), "Support vector domain description," Machine Learning, 54, pp.45-66.

[8] Fawcett, Tom. (2006). "An Introduction to ROC Analysis". Pattern Recognition Letters 27 (8): 861 - 874.

[9] Goldberg, David E., John, H. Holland. (1988). Genetic algorithms and machine learning, Machine learning 3.2 : 95-99.

[10] Lim, Young Won., Lee, Sang Uk. (1990). On the color image segmentation algorithm based on the thresholding and the fuzzy c-means techniques, Pattern recognition 23.9 : 935-952.

[11] LIN Kai-yan., WU Jun-hui., XU Li-hong. (2005). A Survey on Color Image Segmentation Techniques, Journal of Image and Graphics, 10(1) : 1-10.

[12] Mizuno, Koji., et al. (2012). Architectural study of HOG feature extraction processor for real-time object detection, Signal Processing Systems(SiPS), 2012 IEEE Workshop on. IEEE.

[13] Péerez-Cruz, Fernando., et al. (2003). Extension of the nu-svm range for classification, NATO SCIENCE SERIES SUB SERIES 3 COMPUTER AND SYSTEMS SCIENCES 190 : 179-196.

[14] Lee, Donghoon. (2011). Human detection algorithm using PHOG-PLBP feature and intersection Kernel Fuzzy Dual Nu SVM, Dept. of Electrical and Electronic Eng. The Graduate School Yonsei University.

[15] Powers, David. M. W. (2011). Evaluation: From Precision, Recall and FMeasure to ROC, Informedness, Markedness & Correlation(PDF), Journal of Machine Learning Technologies 2 (1): 37 - 63.

[16] Qiang Wu. (2005). SVM Soft Margin Classifiers: Linear Programming versus Quadratic Programming, Massachusetts Institute of Technology, May 2005, Vol.17, No.5, 1160-1187, Posted Online March 13, 2006.

[17] Sheppard, Adrian P., Robert M. Sok, and Holger Averdunk. (2004). Techniques for image enhancement and segmentation of tomographic images of porous materials, Physica A: Statistical mechanics and its applications 339.1 : 145-151.

[18] Steve Gunn. (1998). Support Vector Machine for Classification and Regression, ISIS Technical report, university of Southampton

[19] Vapnik., Vladimir Naumovich., Vlamimir Vapnik. (1998). Statistical learning theory, Vol. 1. New York: Wiley.

[20] Y. Chen., X. Zhou., T.S. Huang. (2001). One-class SVM for learning in image retrieval, Proc. of IEEE International Conference on Image Processing.

Abstract

Support Vector Machine을 이용한 동영상 내 군함 검출

Cho, Chang Sung
Industrial Engineering
The Graduate School
Seoul National University

One could call the modern age an age of artificial intelligence. One such example is automatic recognition of license plate numbers and faces through installing camera. Even the military is replacing it decreaing number of personnel by increasing its number of security cameras. Even the Navy is installing fences with cameras for monitoring islands and the waters. If such equipment are able to differentiate military ships from other objects that come into contact it would significantly help security. Support Vector Machine algorithms are used in order to detect military ships in imagery.

Footage can capture and save images real time. All that needs to be done is determine whether the image is that of a military vessel. Objects that could come into contact out at sea include merchant vessels, ocean liners, fishing vessels, military ships and other floating objects. Accuracy of the SVM test using Histogram of Oriented(HOG) based on edge distribution information revealed that the highest percentage was only a relatively low 62.7%

Most military ships have grey hulls. On the other hand, merchant vessels, cruise ships and fishing vessels are usually painted white, red, blue colors. That is why they have color information as well as HOG inputted in their SVM. The results of accuracy and precision showed 77.2 and 80.6 percent differentiation rate but it takes much time of showing the results.

Maximum value of recall rate showed 81.9 percent of in Random Forest. but precision of Random Forext showed 69.1 percent.

Security is crucial to the military. One mistake made during a security patrol determined the success or failure of a mission of a mission on numerous occasions. Security missions must always guarantee a 100 percent success rate. However tests show only 77.2 and 81.9 percent with a long way to go , a number that is simply unacceptable in SVM and Random Forest algorithms. However, implementation of military ship detection system could improve the success rate.

Key words : Navy ship, SVM, Support Vector Machine, HOG, detection, machine learning

Appendix A

Main.java

```
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.util.ArrayList;
import java.util.List;


public class Main {


    static int[][] Lx = {{0, 1, 0}, {1, -4, 1}, {0, 1, 0}};               // LoG mask Lx
    static int[][] Ly = {{1, 1, 1}, {1, -8, 1}, {1, 1, 1}};               //  LoG mask Ly

    static int[][] Lx = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};             // Sobel mask Lx
    static int[][] Ly = {{1, 0, -1}, {2, 0, -2}, {1, 0, -1}};             // Sobel mask Lx

    static int[][] Lx = {{1, 0, 0}, {0, -1, 0}, {0, 0, 0}};               //   Robert mask Lx
    static int[][] Ly = {{0, -1, 0}, {1, 0, 0}, {0, 0, 0}};               //   Robert mask Ly

    static int[][] Lx = {{-1, -1, -1}, {0, 0, 0}, {1, 1, 1}};             // Prewitt mask Lx
    static int[][] Ly = {{-1, 0, 1}, {-1, 0, 1}, {-1, 0, 1}};             // Prewitt mask Ly




    static int[][] getMatrix(BufferedImage img) {
        int w = img.getWidth();    // 이미지  폭 크기 값
        int h = img.getHeight();  // 이미지 높이 크기 값
        int[][] mat = new int[w][h];  // w, h 값의 mat 배열

        for (int i = 0; i < w; i++) {
            for (int j = 0; j < h; j++) {
               Color c = new Color(img.getRGB(i, j));                  // c는 i,j번째의 RGB의 값
               int v = (c.getRed() + c.getGreen() + c.getBlue()) / 3;     // v는 RGB의 평균 값
               mat[i][j] = v;      // mat 배열 : i, j번째 RGB의 평균값을 각각 저장
            }
        }
        return mat;
    }


    static int[][] getGradient(int[][] input, int[][] filter){
        int w = input.length;
        int h = input[0].length;
        int[][] output = new int[w][h];
        for (int i = 1; i < w-1; i++) {
            for (int j = 1; j < h-1; j++) {
               int v = filter[0][0] * input[i-1][j-1] + filter[0][1] * input[i-1][j] + filter[0][2] * input[i-1][j+1]
                     + filter[1][0] * input[i][j-1] + filter[1][1] * input[i][j] + filter[1][2] * input[i][j+1]
                     + filter[2][0] * input[i+1][j-1] + filter[2][1] * input[i+1][j] + filter[2][2] *input[i+1][j+1];

               output[i][j] = v;
            }
        }
        return output;
    }

    static int[][] getMagnitude(int[][] ix, int[][] iy){
        int w = ix.length;
        int h = ix[0].length;
        int[][] output = new int[w][h];

        for (int i = 1; i < w-1; i++) {
            for (int j = 1; j < h-1; j++) {
               int v = (int) Math.sqrt((double)(ix[i][j] * ix[i][j] + iy[i][j] * iy[i][j]));
               output[i][j] = v;
            }
        }
        return output;
    }
```

```
static int[][] getTheta(int[][] ix, int[][] iy){
      int w = ix.length;
      int h = ix[0].length;
      int[][] output = new int[w][h];
          output[i][j]
      for (int i = 1; i < w-1; i++) {
            for (int j = 1; j < h-1; j++) {
                double radian = Math.atan(iy[i][j] / (ix[i][j] == 0 ? 0.001 : (double) ix[i][j]));
                = (int) Math.toDegrees(radian);
            }
      }
      return output;
}


static double[][][] createHistograms(int[][] g, int[][] theta, int cellsize) {
      int w = g.length;
      int h = g[0].length;
      int nWCell = w / cellsize;
      int nHCell = h / cellsize;
      int nBin = 9;

      double[][][] histograms = new double[nWCell][nHCell][nBin];
      for (int wCellIdx = 0; wCellIdx < nWCell; wCellIdx++) {
            for (int hCellIdx = 0; hCellIdx < nHCell; hCellIdx++) {
                  for (int i=0; i<cellsize; i++) {
                        for (int j=0; j<cellsize; j++) {
                              int x = cellsize * wCellIdx + i;
                              int y = cellsize * hCellIdx + j;

                              int bin = (theta[x][y] + 90) / (180 / nBin);
                              histograms[wCellIdx][hCellIdx][bin] += g[x][y];

                        }
                  }
            }
      }
      return histograms;
}

static void globalNormalization(double[][][] histograms) {
      double maxv = 0.001;
      for (int i=0; i<histograms.length; i++) {
            for (int j=0; j<histograms[i].length; j++) {
                  for (int z=0; z<histograms[i][j].length; z++) {
                        double v = histograms[i][j][z];
                        maxv = Math.max(v, maxv);
                  }
            }
      }

      double normalizationRate = 1.0 / maxv;
      for (int i=0; i<histograms.length; i++)
            for (int j=0; j<histograms[i].length; j++)
                  for (int z=0; z<histograms[i][j].length; z++)
                        histograms[i][j][z] *= normalizationRate;
                        return;
}


static double[] num_column_Gray_Value(BufferedImage img){
      int w = img.getWidth();    // 이미지  폭 크기 값
      int h = img.getHeight();   // 이미지 높이 크기 값
      int[][] eachRed = new int[w][h];
      int[][] eachGreen = new int[w][h];
      int[][] eachBlue = new int[w][h];
      int[][] r_g = new int[w][h];
      int[][] g_b = new int[w][h];
      int[][] b_r = new int[w][h];
      double[] column_gray_Value = new double[w];

      for (int i=0; i<w; i++){
            int column_Gray_Value = 0;
            int update_column_Gray_Value = 0 ;
            for (int j=0; j<h; j++){
                  Color c = new Color(img.getRGB(i, j));
                  eachRed[i][j]=c.getRed();
                  eachGreen[i][j]=c.getGreen();
```

```
                eachBlue[i][j]=c.getBlue();
                r_g[i][j] = Math.abs(eachRed[i][j]-eachGreen[i][j]);
                g_b[i][j] = Math.abs(eachGreen[i][j]-eachBlue[i][j]);
                b_r[i][j] = Math.abs(eachBlue[i][j]-eachRed[i][j]);

                if(eachRed[i][j]<30 && eachGreen[i][j]<30 && eachBlue[i][j]<30 ){
                    update_column_Gray_Value = column_Gray_Value;
                }else if(eachRed[i][j]>220 && eachGreen[i][j]>220 && eachBlue[i][j]>220 ){
                    update_column_Gray_Value = column_Gray_Value;
                }else if(r_g[i][j]<20 && g_b[i][j]<20 && b_r[i][j]<20  ){
                    column_Gray_Value = column_Gray_Value + 1 ;
                    update_column_Gray_Value = column_Gray_Value;
                }else update_column_Gray_Value = column_Gray_Value;
            } column_gray_Value[i] = update_column_Gray_Value;

        } return column_gray_Value;
    }

    static double[] num_column_White_Value(BufferedImage img){
        int w = img.getWidth();   // 이미지  폭 크기 값
        int h = img.getHeight();  // 이미지 높이 크기 값
        int[][] eachRed = new int[w][h];
        int[][] eachGreen = new int[w][h];
        int[][] eachBlue = new int[w][h];
        double[] column_White_Value = new double[w];

        for (int i=0; i<w; i++){
            int White_Value = 0;
            int update_column_White_Value = 0 ;
            for (int j=0; j<h; j++){
                Color c = new Color(img.getRGB(i, j));
                eachRed[i][j]=c.getRed();
                eachGreen[i][j]=c.getGreen();
                eachBlue[i][j]=c.getBlue();
                if(eachRed[i][j]>220 && eachGreen[i][j]>220 && eachBlue[i][j]>220){
                    White_Value = White_Value + 1 ;
                    update_column_White_Value = White_Value;
                }
            } column_White_Value[i] = update_column_White_Value;
        }
         return column_White_Value;
    }

    static double[] num_column_Blue_Value(BufferedImage img){
        int w = img.getWidth();   // 이미지  폭 크기 값
        int h = img.getHeight();  // 이미지 높이 크기 값
        int[][] eachRed = new int[w][h];
        int[][] eachGreen = new int[w][h];
        int[][] eachBlue = new int[w][h];
        double[] column_Blue_Value = new double[w];

        for (int i=0; i<w; i++){
            int Blue_Value = 0;
            int update_column_Blue_Value = 0 ;
            for (int j=0; j<h; j++){
                Color c = new Color(img.getRGB(i, j));
                eachRed[i][j]=c.getRed();
                eachGreen[i][j]=c.getGreen();
                eachBlue[i][j]=c.getBlue();
                if(eachRed[i][j]<90 && eachGreen[i][j]<90 && eachBlue[i][j]>150  ){
                    Blue_Value = Blue_Value + 1 ;
                    update_column_Blue_Value = Blue_Value;
                }
            } column_Blue_Value[i] = update_column_Blue_Value;
        }
        return column_Blue_Value;
    }
```

```
	static double[] num_column_Red_Value(BufferedImage img){
		int w = img.getWidth();    // 이미지  폭 크기 값
		int h = img.getHeight();  // 이미지 높이 크기 값
		int[][] eachRed = new int[w][h];
		int[][] eachGreen = new int[w][h];
		int[][] eachBlue = new int[w][h];
		double[] column_Red_Value = new double[w];

		for (int i=0; i<w; i++){
			int Red_Value = 0;
			int update_column_Red_Value = 0 ;
			for (int j=0; j<h; j++){
				Color c = new Color(img.getRGB(i, j));
				eachRed[i][j]=c.getRed();
				eachGreen[i][j]=c.getGreen();
				eachBlue[i][j]=c.getBlue();
				if(eachRed[i][j]>150 && eachGreen[i][j]<90 && eachBlue[i][j]<90   ){
					Red_Value = Red_Value + 1 ;
					update_column_Red_Value = Red_Value;
				}
			} column_Red_Value[i] = update_column_Red_Value;
		}
		return column_Red_Value;
	}

	static double[] processImage(BufferedImage oriImage) {
		int[][] mat = getMatrix(oriImage);
		int[][] sobelx = getGradient(mat, Lx); // i_x 행렬 추출
		int[][] sobely = getGradient(mat, Ly); // i_y 행렬 추출
		int[][] g = getMagnitude(sobelx, sobely); // G 행렬 추출
		int[][] theta = getTheta(sobelx, sobely); // theta 행렬 추출
		double[][][] hog = createHistograms(g, theta, 6); // hog 피쳐 추출
		globalNormalization(hog); // hog 에 대해 global normalization 수행
		double[] column_gray_Value = num_column_Gray_Value(oriImage);  // 열 Gray 값 배열 추출
		double[] column_white_Value = num_column_White_Value(oriImage);  // 열 white 값 배열 추출
		double[] column_red_Value = num_column_Red_Value(oriImage);  // 열 Red 값 배열 추출
		double[] column_blue_Value = num_column_Blue_Value(oriImage);  // 열 Blue 값 배열 추출

		List<Double> featList = new ArrayList<Double>();
		for (int i=0; i< hog.length; i++)
			for (int j=0; j<hog[i].length; j++)
				for (int k=0; k<hog[i][j].length; k++)
					featList.add(hog[i][j][k]);

		for (int i=0; i< column_gray_Value.length; i++)
			featList.add(column_gray_Value[i]);
		for (int i=0; i< column_white_Value.length; i++)
			featList.add(column_white_Value[i]);
		for (int i=0; i< column_blue_Value.length; i++)
			featList.add(column_blue_Value[i]);
		for (int i=0; i< column_red_Value.length; i++)
			featList.add(column_red_Value[i]);

		double[] featVec = new double[featList.size()];
		for (int i=0; i<featList.size(); i++)
			featVec[i] = featList.get(i);
		return featVec;
	}
}
```

Appendix B

MyClassifier.java

```
import java.awt.image.BufferedImage;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.functions.Logistic;
import weka.classifiers.functions.LibSVM;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.RandomForest;
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import java.util.Calendar;

public class MyClassifier {

    static void toCSV(List<double[]> X, List<Boolean> y, String csvpath) throws IOException {
        int nAtts = X.get(0).length;
        BufferedWriter fw = new BufferedWriter(new FileWriter(csvpath));
        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < nAtts; i++)
            sb.append("f" + i + ",");
        sb.append("theClass\n");
        fw.write(sb.toString());

        for (int idx = 0; idx < X.size(); idx++) {
            double[] x = X.get(idx);
            sb = new StringBuilder();
            for (int i = 0; i < nAtts; i++)
                sb.append(x[i] + ",");
            sb.append((y.get(idx) ? "pos" : "neg") + "\n");
            fw.write(sb.toString());
        }

    fw.flush();
    fw.close();
}

    static Instances toInstances(List<double[]> X, List<Boolean> y) {
        // Declare two numeric attributes
        int nAtts = X.get(0).length;
        List<Attribute> attList = new ArrayList<Attribute>();
        for (int i = 0; i < nAtts; i++)
            attList.add(new Attribute("f" + i));

        // Declare the class attribute along with its values
        FastVector fvClassVal = new FastVector(2);
        fvClassVal.addElement("pos");
        fvClassVal.addElement("neg");
        Attribute classAtt = new Attribute("theClass", fvClassVal);

        // Declare the feature vector
        FastVector fvWekaAttributes = new FastVector(nAtts + 1);
        for (Attribute att : attList)
             fvWekaAttributes.addElement(att);
        fvWekaAttributes.addElement(classAtt);
```

```java
        // Create an empty dataset
        Instances instances = new Instances("Relation", fvWekaAttributes, X.size());
        // Set class index
        instances.setClassIndex(nAtts);

        for (int idx = 0; idx < X.size(); idx++) {
            double[] x = X.get(idx);
            // Create the instance
            Instance inst = new Instance(nAtts + 1);

            for (int i = 0; i < x.length; i++)
                 inst.setValue((Attribute) fvWekaAttributes.elementAt(i), x[i]);
            inst.setValue((Attribute) fvWekaAttributes.elementAt(nAtts), y.get(idx) ? "pos" : "neg");
            // add the instance
            instances.add(inst);
        }

        return instances;
    }

    static void listFiles(File dir, List<File> files) {
        for (File file : dir.listFiles()) {
            if (file.isDirectory())
                istFiles(file, files);
            else
                files.add(file);
        }
    }

    static Instances toInstances(String posDirPath, String negDirPath, String csvpath) throws Exception {
        List<File> posFiles = new ArrayList<File>();
        listFiles(new File(posDirPath), posFiles);

        List<File> negFiles = new ArrayList<File>();
        listFiles(new File(negDirPath), negFiles);

        System.out.println(posFiles.size() + " pos, " + negFiles.size() + " negs.");

        List<double[]> X = new ArrayList<double[]>();
        List<Boolean> y = new ArrayList<Boolean>();

        for (File f : posFiles) {
            BufferedImage img = ImageIO.read(f);
            double[] featVec = Main.processImage(img);
            X.add(featVec);
            y.add(true);
        }
        for (File f : negFiles) {
            BufferedImage img = ImageIO.read(f);
            double[] featVec = Main.processImage(img);
            X.add(featVec);
            y.add(false);
        }
        System.out.println("extract hog features.");

        Instances dataset = null;
        if (csvpath == null) {
             dataset = toInstances(X, y);
        } else {
            toCSV(X, y, csvpath);
            System.out.println("save instances to csv file.");

            DataSource source = new DataSource(csvpath);
            dataset = source.getDataSet();
            if (dataset.classIndex() == -1)
                dataset.setClassIndex(dataset.numAttributes() - 1);
        }

        System.out.println("changed to weka instances.");
        return dataset;
    }

    public String formatTime(long lTime){
            Calendar c = Calendar.getInstance();
            c.setTimeInMillis(lTime);
            return(c.get(Calendar.HOUR_OF_DAY)+"시" +
            c.get(Calendar.MINUTE)+"분"+c.get(Calendar.SECOND)+"."+ c.get(Calendar.MILLISECOND)+"초");
    }
```

```
	public static void main(String[] args) throws Exception {
		long startTime = System.currentTimeMillis();
		String trPosDirPath = "C:/Users/Sung_Young/Desktop/data/600.training_set_MS";
		String trNegDirPath = "C:/Users/Sung_Young/Desktop/data/600.training_set_ETC";
		String trCSVPath = "resources/csv/training.csv";

		Instances trainingSet = toInstances(trPosDirPath, trNegDirPath, trCSVPath);

		String tePosDirPath = "C:/Users/Sung_Young/Desktop/data/600.test_set_MS";
		String teNegDirPath = "C:/Users/Sung_Young/Desktop/data/600.test_set_ETC";
		String teCSVPath = "resources/csv/test.csv";

		Instances testSet = toInstances(tePosDirPath, teNegDirPath, teCSVPath);

//		Create a classifier
		Classifier clf = new NaiveBayes();
//		Classifier clf = new J48();
//		Classifier clf = new LibSVM();
//		Classifier clf = new RandomForest();
//		Classifier clf = new Logistic();

		clf.buildClassifier(trainingSet);

		// serialize model
		String modelpath = "resources/models/NaiveBayes.model";
		ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(modelpath));

		oos.writeObject(clf);
		oos.flush();
		oos.close();

		System.out.println();
		Classifier savedClf = (Classifier) weka.core.SerializationHelper.read(modelpath);
		System.out.println("load saved model from " + modelpath);

		Evaluation evalCls = new Evaluation(testSet);
		evalCls.evaluateModel(savedClf, testSet);

		double[][] confusionMat = evalCls.confusionMatrix();
		double TPR = confusionMat[0][0] / (confusionMat[0][0]+confusionMat[0][1]);
		double Precision = confusionMat[0][0] / (confusionMat[0][0]+confusionMat[1][0]);
		double TNR = confusionMat[1][1] / (confusionMat[1][0]+confusionMat[1][1]);
		System.out.println("TP : " + confusionMat[0][0] + " " +  "FN : " + confusionMat[0][1]);
		System.out.println("FP : " + confusionMat[1][0] + " " +  "TN : " + confusionMat[1][1]);
		System.out.println("참긍정률_Recall(TP/(TP+FN)) : " + TPR +"(군함을 군함이라고 한 것)");
		System.out.println("정밀도_Precision_(TP/(TP+TN)) : " + Precision +"(군함이라고 한 것 중 진짜
		군함인 것)");
		System.out.println("참률_Recall(TP/(TP+FN)) : " + TNR +"(군함이 아닌 것을 군함이 아니라고 한 것)");
		System.out.println(evalCls.toSummaryString());
		long endTime = System.currentTimeMillis();
		System.out.println("## 시작시간 : " + new MyClassifier().formatTime(startTime));
		System.out.println("## 종료시간 : " + new MyClassifier().formatTime(endTime));

		float a = (endTime - startTime) / 1000.0f;
		int fS = (int) a;
		int H;
		int M;
		int fM;
		int S;
		if(fS<60){
			H = 0 ;
			M = 0;
			S = fS;
			System.out.println("## 소요시간 : " + H + "시" + M + "분" +  S + "초");
		}else if(fS>=60 && fS<3600){
			H = 0 ;
			M = fS	/ 60 ;
			S = fS -60*M;
			System.out.println("## 소요시간 : " + H + "시" + M + "분" +  S + "초");
		}else if(fS>=3600){
			H = fS / 3600;
			fM = fS - 3600*H;
			if(fM>=60){
				M = fM / 60 ;
				S = fM - 60*M ;
				System.out.println("## 소요시간 : " + H + "시" + M + "분" +  S + "초");
			}else if(fM<60){
				M = 0;
```

```
					S = fM;
					System.out.println("## 소요시간 : " + H + "시" + M + "분" +  S + "초");
				}

			}
		}
}
```