ISSN: 1226-7244 (Print)
ISSN: 2288-243X (Online)
논문번호 16-04-10
62

j.inst.Korean.electr.electron.eng.Vol.20,No.4,398~411,December 2016
http://dx.doi.org/10.7471/ikeee.2016.20.4.398

기계학습 및 분류를 위한 SVM 엔진의 FPGA 구현

FPGA Implementation of SVM Engine for Training and Classification

나 원 섭 *, 정 용 진*★

Wonseob Na*, Yongjin Jeong*★

Abstract

SVM, a machine learning method, is widely used in image processing for it's excellent generalization performance. However, to add other data to the pre-trained data of the system, we need to train the entire system again. This procedure takes a lot of time, especially in embedded environment, and results in low performance of SVM. In this paper, we implemented an SVM trainer and classifier in an FPGA to solve this problem. We parlallelized the repeated operations inside SVM and modified the exponential operations of the kernel function to perform fixed point modelling. We implemented the proposed hardware on Xilinx ZC 706 evaluation board and used TSR algorithm to verify the FPGA result. It takes about 5 seconds for the proposed hardware to train 2,000 data samples and 16.54ms for classification for 1360 X 800 resolution in 100MHz frequency, respectively.

요 약

기계학습 방법의 하나인 SVM은 뛰어난 일반화 성능으로 영상처리 분야에서 많이 사용하고 있다. 하지만 SVM을 이용한 시스템에서 미리 학습된 데이터가 아닌 다른 데이터를 이용하려하면 새로 학습을 시켜야 하는 경우가 생긴다. 특히, 임베디드 환경에서는 이러한 상황에서 학습 시간이 오래 걸려 SVM을 적절히 이용하지 못하는 경우가 있다. 본 논문에서는 이러한 문제점을 해결하기 위하여 SVM의 학습 및 분류를 모두 수행할 수 있도록 하나의 FPGA로 구현하였다. SVM 연산의 복잡성으로 인해 생기는 반복연산을 병렬처리를 통하여 해결하고 커널 사용으로 생기는 지수 연산을 변형하여 고정 소수점 연산이 가능하도록 하였다. 제안하는 하드웨어는 Xilinx사의 ZC 706보드에 구현하였고, 구현한 FPGA의 검증을 위하여 TSR 알고리즘을 이용하였다. 구현한 하드웨어는 100 MHz의 주파수로 동작하며, 2천개의 데이터를 이용한 학습 시 약 5sec가 소요되고 1360 X 800 해상도에서 분류 시 약 16.54msec가 소요됨을 확인했다.

Key words : Machine Learning, SVM, FPGA, Training, Classification

* Dept. of Electronics and Communications Engineering, Kwangwoon University
★ Corresponding author
yjjeong@kw.ac.kr / 02-940-5551
※ Acknowledgment
This work was supported in part by the Research Grant of Kwangwoon University in 2015 and the IT R & D program of Ministry of Trade, Industry and Energy (10049192, Development of a Smart Automotive ADAS SW-Soc for a Self-Driving Car)
Manuscript received Nov. 30, 2016; revised Dec. 26, 2016 ; accepted Dec. 28, 2016

Ⅰ. 서론

기계학습이란 다양한 데이터를 이용한 학습을 통하여 컴퓨터가 스스로 최적의 판단이나 예측을 가능하게 해주는 기술을 말한다. 기계학습은 마케팅, 금융권 등에서 사용하는 데이터마이닝 분야나 숫자인식, 동작인식, 지문인식 등을 수행하는 영상처리 분야 등 다양한 분야에 사용되고 있다.[1] 그 중 SVM(Support Vector Machine)은 단순히 오인식을 줄이기위한 학습을 하는 인공신경망(Artificial Neural Network)와 달리 일반화 성능을 극대화 하는 학습을 하여 영상처리를 통한 패턴인식 분야에서 뛰어난 성능을 나타낸다.[2]

기계학습을 통한 영상처리는 크게 학습(Training)과 분류(Classification)의 두 단계로 나누어진다. 학습 단계는 학습하기 전 찾고자하는 대상의 영상과 찾고자하지 않는 대상의 영상을 특징점 추출기를 이용하여 데이터를 저장하는 과정과 그 데이터를 이용하여 일정한 방식의 학습방법을 통해 데이터의 종류를 구분하는 과정으로 나누어진다. 여기서 특징점 추출기란 물체의 특징이 될 수 있는 코너부분이나 선분영역을 찾아내어 일정한 형태로 저장하는 것을 말한다. 특징점 추출기의 종류로는 HOG(Histogram of Oriented Gradient), SIFT(Scale Invariant Feature Transform), SURF(Speeded Up Robust Feature) 등이 있다. 특징점 추출기를 통해 찾아낸 데이터는 각 추출기별로 정해져 있는 일정한 형태의 기술자(descriptor)로 저장된다. 학습을 하기 위해서는 동일한 특징점 추출기를 이용하여 positive 영상(찾고자하는 영상)과 negative 영상(찾고자하지 않는 영상)의 특징점을 추출한다. 그 후 추출한 특징점을 이용하여 일련의 방식으로 학습을 진행한다. 이렇게 학습된 데이터는 일정한 형태로 저장되어 데이터의 분류에 사용된다. 학습에 사용하는 영상의 수가 많을수록 그 정확도는 올라가나 일정수치 이후는 단순히 영상의 수를 늘리는 것만으로는 정확도를 크게 높이지 못한다. 또한 학습 시 보유한 positive 영상의 80%는 학습에 사용하고 나머지 20%의 영상은 학습이 제대로 되었는지 확인하는데 사용한다.

분류 단계는 전체 입력 영상 중 찾고자 하는 대상이 있을만한 후보 영역을 분할한 후 후보 영역을 학습에 사용한 특징점 추출기를 이용하여 특징점을 추출한다. 이렇게 추출한 특징점 데이터와 학습의 결과로 나온 데이터를 일련의 방식을 통해 연산하면 그 데이터가 positive인지 negative인지 판별할 수 있다. 이렇게 판별한 데이터는 positive일 경우 입력영상에 일정한 표시를 하여 찾고자하는 대상을 표시한다.

일반적으로 학습과정은 분류기를 사용하는 환경이 아닌 고성능의 PC를 이용하여 분류과정을 수행하기 전 미리 진행한다. 하지만 분류과정 중 새로운 positive 데이터가 나오면 다시 학습을 진행해야 하는데 학습기와 분류기의 사용 환경이 달라 즉각적인 학습이 이루어지기 어렵다. 또한, SVM 학습 및 분류는 MAC(Multiply-ACcumulate) 연산을 반복적으로 수행하기 때문에 임베디드 환경에서 소프트웨어만으로 수행 시 원할한 동작의 어려움이 있다.

따라서 본 논문에서는 SVM 학습과 분류를 모두 수행할 수 있는 통합 SVM 시스템의 하드웨어 가속기를 제안한다. 연산 부하가 많이 걸리는 MAC 연산을 병렬 처리를 통하여 개선하고 커널(kernel) 함수의 사용으로 생기는 지수연산을 테일러급수를 이용하여 고정소수점 연산이 가능하도록 식을 변형하였다. SVM 학습에는 SMO(Sequential Minimal Optimization) 방식을 이용하여[3] 연산의 부하를 줄이고 학습한 결과를 비교 하기위해 오픈소스인 LibSVM을 사용하였다.[4] 설계한 시스템의 검증을 위해 TSR(Traffic Sign Recognition) 알고리즘에 적용하였다.

본 논문의 구성으로는 2장에서는 제안하는 하드웨어와 관련한 연구들을 소개하고 3장에서는 SVM 알고리즘에 대해 설명한다. 4장에서는 설계한 하드웨어 모듈에 대한 내용을 소개하며 5장에서는 TSR 알고리즘을 이용하여 제안하는 하드웨어의 성능을 분석한다.

Ⅱ. 관련연구

이번 장에서는 앞서 진행되었던 SVM 알고리즘의 하드웨어 연구에 관해 검토해본다. 본 논문에서 제안하는 하드웨어는 크게 학습부분과 분류부분으로 나눌 수 있고 각 부분을 연구한 논문들에 대해 살펴보도록 한다.

1. SVM 학습(Training)

SVM 학습을 하드웨어로 구현한 연구는 크게 두 가지로 나누어 볼 수 있다. 한 가지는 SVM 학습에서 가장 연산부하가 많이 걸리는 부분인 MAC 연산을 하드웨어로 구현하여 연산부하를 줄이는 방법[5]이고 다른 한 가지는 SVM 학습과정 전체를 하드웨어로 구현하는 방법[6]이다. MAC 연산만을 하드웨어로 구현한 [5]의 경우 학습을 위해서는 호스트 PC가 데이터 전송 및 전체 과정의 컨트롤을 담당한다. 이렇게 학습과정의 일부만을 하드웨어로 구현한 경우 구현이 간단하다는 장점이 있을 수 있으나 하드웨어로 동작하는 부분을 제외한 부분은 호스트 PC에 의존하기 때문에 임베디드 환경에서 사용 시 좋은 성능을 기대하기 어렵다는 단점이 생긴다. 이와는 달리 [6]에서는 전체 학습과정을 하드웨어로 구현하여 호스트 PC를 사용하지 않고 단독으로 SVM 학습을 진행 할 수 있어 호스트 PC에 관계없이 좋은 성능을 기대 할 수 있다. 하지만 [6]의 경우는 적은 하드웨어 사용량을 목적으로 설계를 하여 성능 향상이 크지 않고 반복횟수를 제한하고 Linear 커널을 사용하여 학습 결과를 이용할 시 정확도가 떨어진다는 단점이 있다.

본 논문에서는 [6]와 같이 SVM 학습과정 전체를 하드웨어로 구현하면서 임베디드 환경에서도 원할히 사용할 수 있도록 수행시간을 단축시켰다. 또한 학습 결과의 정확도를 높이기 위하여 RBF(Radial Basis Function) 커널을 이용하고 반복횟수의 제한을 두지 않았다.

2. SVM 분류(Classification)

SVM 분류를 하드웨어로 구현하는데 있어 가장 큰 목표는 실시간 처리를 통하여 고속 시스템에서도 사용이 가능하도록 하는 것이다. SVM 분류를 실시간으로 처리하기 위하여 여러 가지 가속화 방법들이 제안되었다.[7]-[12] 첫 번째 방법으로는 SVM 분류 시 사용하는 RBF 커널의 내부 MAC 연산 구조를 변형한 Hardware-Friendly 커널을 사용하여 가속화를 하였다.[8][9] 하지만 해당 논문에서 사용한 커널은 RBF 커널에 비해 분류 성능이 낮은 단점이 있다. 두 번째 방법은 GPU(Graphic Processing Unit)를 이용하여 SVM 분류를 가속화 하는 것이다.[10][12] 그런데 GPU의 경우 FPGA(Field-Programmable Gate Arrays)보다 최적화가 어렵고 전력 사용량이 높다는 문제가 있어 임베디드 시스템에서 사용하기에는 적합하지 않다. 마지막 방법은 FPGA를 이용하여 SVM 분류기 내부의 MAC 연산을 병렬적으로 처리하는 것이다.[7][11]

본 논문에서는 [7],[11]과는 달리 실시간 처리를 하는데 제한이 되는 반복 연산을 분류기 내부의 MAC 연산을 병렬 처리 하는 것이 아닌 SVM 분류기 자체를 병렬 처리 하였다.[13] 분류에는 학습과 같이 RBF 커널을 이용하였고 설계한 분류기를 검증하기 위해 TSR 알고리즘을 이용하였다.

III. Support Vector Machine

1. SVM 학습

SVM은 1995년 V. Vapnik이 제안한 알고리즘이다.[14] 1장에서 서술한 바와 같이 SVM은 인공신경망과는 달리 일반화 능력을 극대화하기 위한 학습을 진행하는데 그림 1은 SVM 학습 과정에서 일반화 능력을 높이기 위한 과정을 도식화 한 것이다.[15]

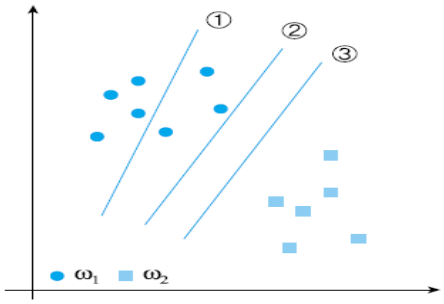

Fig. 1. SVM Training Process
그림 1. SVM 학습과정

그림 1의 ①은 오류가 3개 있는 분류이다. 인공신경망은 초기 값 ①에서 시작하여 오류를 줄이며 오류가 없어지는 ②를 찾아간 후 학습을 종료한다. 하지만 SVM은 ②에서 멈추지 않고 ③을 찾아내어 학습을 종료하는데, 이 둘은 미지의 패턴을 분류하는 일반화 능력에서 상당한 차이를 보이게 된다. 즉, ③의 분류기가 테스트 패턴의 변형에도 더 강인한 분류기이고 이는 분류기의 성능이 더 뛰어나다는 것을 의미하게 된다.

SVM에서 학습을 통해 결정되는 분류기준을 초

평면(Hyperplane)이라고 하고 초평면과 가장 가까이 있는 샘플들을 서포트 벡터(Support Vector)라 한다. 그리고 서포트 벡터와 초평면 사이 거리의 두 배를 마진(Margin)이라 부른다. 즉, SVM 학습은 마진이 최대가 되는 초평면을 찾아가는 과정이고 기본적으로 SVM은 선형 이진 분류기의 특성을 가지게 된다. 그림 2는 두 종류의 데이터를 이용하여 SVM 학습을 하였을 때의 예시를 보여준다.

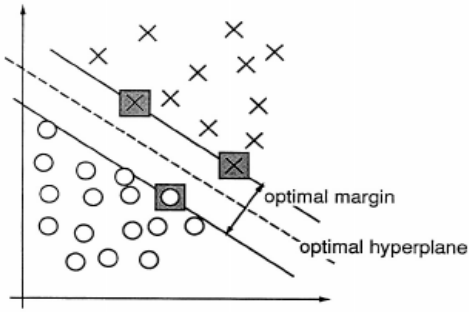

Fig. 2. Example of SVM Training
그림 2. SVM 학습의 예

하지만 일반적인 학습 데이터는 비선형적인 특성을 가지는 경우가 많은데 이를 해결하기 위해 SVM은 커널기법을 이용한다. 커널기법은 일정한 함수를 통해 평면상으로 보면 비선형적인 데이터를 더 높은 차원으로 매핑(mapping)하여 선형적인 형태가 나오도록 하는 방법을 말한다. 그림 3은 커널을 사용하였을 때 학습데이터가 고차원으로 매핑되는 것을 도식화 한 것이다.[16]

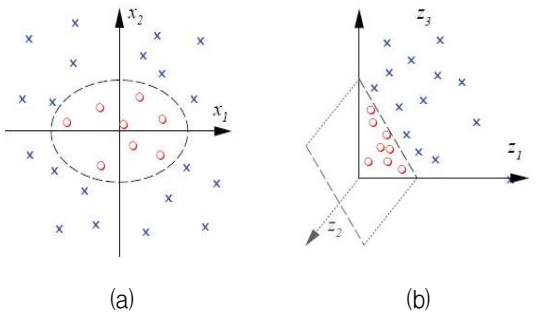

Fig. 3 Example of Kernel Method
그림 3. 커널기법의 예

그림 3-(a)는 커널기법을 사용하기 전 선형 분류가 되지 않는 상황이며 그림 3-(b)는 커널기법을 사용하여 선형분류가 가능한 차원으로 매핑이 된 것을 나타낸다. SVM 학습에는 다양한 커널들이 사용 가능한데 표 1은 각 SVM에서 주로 사용하는 커널 함수를 보여준다.

세 커널 함수 중에서 가장 뛰어난 분류 성능을 나타내는 것은 RBF지만, Linear 및 Polynomial 커널에서는 벡터 내적을 하는 반면 RBF 커널의 경우 벡터 norm 연산을 하기 때문에 더 많은 계산량을 필요로 한다.[12]

Table 1. Kernel Functions
표 1. 커널 함수

Kernel Function	Formula
Linear	$k(x,v) = x \cdot v$
Polynomial	$k(x,v) = (1+x \cdot v)^d$
RBF	$k(x,v) = \exp(-a\|x-v\|^2)$

최적의 초평면을 찾기 위한 학습을 위해서는 식 (1)의 연산을 수행하여야 한다.[15]

$$\max_{\alpha}(\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(x_i,x_j)) \quad (1)$$
$$s.t\sum_{i=1}^{N}\alpha_i y_i = 0, 0 \le \alpha_i \le C, i = 1,2,\cdots,N$$

여기서 x는 입력 데이터이고 y는 입력 데이터에 따른 값으로 1 또는 -1의 값을 가진다. α는 라그랑제 승수를 의미하고 C는 완전한 선형 분리가 이루어지지 않을 때 허용할 수 있는 오차를 나타내는 값이다. 그리고 $K(x_i,x_j)$는 커널함수를 의미하며 N은 입력 데이터의 숫자이다.

식 (1)에서 확인할 수 있듯이 SVM 학습을 하기 위해서는 α에 대한 2차 함수 최대화 문제를 풀어내야 한다. 하지만 N이 커지게 되면 2차 항의 수가 $\frac{N(N+1)}{2}$개가 생기게 되어 쉽게 풀어내기 어려워진다. 이러한 문제를 해결하기 위해 모든 변수를 한번에 계산하는 것이 아닌 일련의 방법으로 선택된 적은 수의 데이터를 이용하여 학습을 한 후 그것을 바탕으로 다시 데이터를 선택하여 학습을 반복하는 방식으로 SVM 학습을 진행하게 된다. 이러한 방법은 크게 Chunking[17], Osuna[18], 그리고 SMO[3] 등이 있는데

Chunking은 일정수의 데이터를 이용하여 학습을 하여 서포트 벡터의 후보군을 남기고 처음 학습을 시작했을 때와 같은 수의 데이터와 같이 다시 학습을 반복하여 최종적으로 학습결과를 내는 방식이다. Osuna는 Chunking과 비슷하게 일정량의 데이터를 이용하여 학습을 하여 서포트 벡터의 후보군을 찾아내지만 Chunking과는 달리 서포트 벡터의 후보로 제외된 수만큼 데이터를 추가하여 학습을 진행해간다. 즉, Osuna는 Chunking보다 메모리 사용량을 획기적으로 줄인 알고리즘이라고 할 수 있다. SMO는 학습을 진행하는데 있어 가장 적은 수인 두 개의 데이터를 이용하여 학습을 진행하고 각종 파라메터를 업데이트 한 뒤 다시 새로운 두 데이터를 이용하여 학습을 진행하는 방법이다. SMO는 Chunking과 Osuna에 비해 속도 및 메모리 사용량에서 많은 이득을 볼 수 있어 현재 가장 많이 사용하는 방법이다.[3][15] 그림 4는 Chunking, Osuna, 그리고 SMO가 사용하는 데이터 수 및 메모리 사용량을 도식화하여 나타낸 것이다.

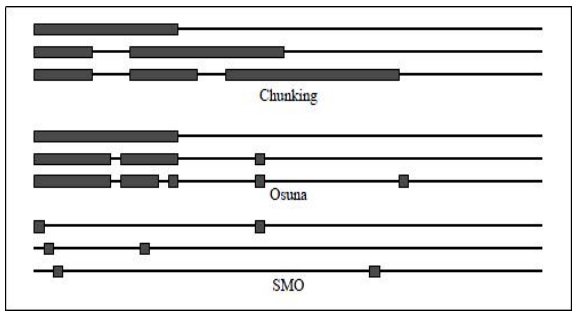

Fig. 4. Methods of SVM Training[3]

그림 4. SVM 학습의 방법

그림 4에서 얇은 검은 선은 전체 학습데이터를 나타내고 굵은 검은 선은 식 (1)을 풀어내기 위해 사용하고 있는 샘플 데이터를 나타낸다. Chunking의 경우는 사용하는 샘플 데이터의 수가 계속해서 증가하고 Osuna의 경우는 거의 일정한 크기의 샘플 데이터를 이용하여 학습을 진행하는 것을 확인할 수 있다. 반면에 SMO는 적은 샘플 데이터를 이용하여 학습을 진행하여 메모리 사용량이 다른 방법에 비해 획기적으로 감소하였음을 확인할 수 있다.

2. SVM 분류

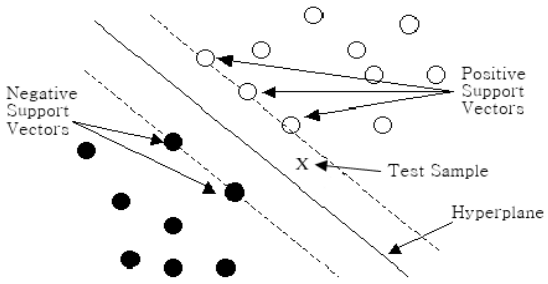

Fig. 5. Example of SVM Classification

그림 5. SVM 분류의 예

학습을 통해 만들어진 초평면을 이용하여 임의의 샘플이 입력되었을 때 그 샘플이 positive 영역에 속하는지 negative 영역에 속하는지 판단하는 것을 분류라 한다. 그림 5는 SVM을 이용한 분류를 도식화 한 것이다. SVM을 이용한 분류는 커널기법의 사용여부에 따라 방법이 바뀌게 된다. 커널기법을 사용하지 않은 선형 SVM의 경우는 학습을 통해 결정된 초평면의 법선 벡터를 미리 구해놓고 테스트 샘플과의 벡터 내적(inner product, dot product)값과 바이어스 값의 합으로 분류를 할 수 있지만 커널기법을 사용한 비선형 SVM의 경우 초평면의 법선 벡터를 미리 구해놓을 수 없고 커널함수 연산을 서포트 벡터의 수만큼 반복하여 수행하여야 한다.[15][16] 즉, 비선형 SVM의 경우는 서포트 벡터의 수에 비례하여 분류시간이 증가하게 된다.

일반적으로 학습 데이터가 증가하면 서포트 벡터의 수도 증가한다. 이로 인해 분류 시간이 증가하는 상황이 발생하여 실시간 처리를 하는데 저해하게 된다. 하지만 이와는 달리 입력 영상의 해상도가 높아지게 되면 분류해야 하는 개체가 늘어나 분류 시간이 증가하게 되는 경우가 발생한다. 즉, SVM 분류의 실시간 처리를 위해서 다수의 서포트 벡터로 인해 생기는 반복 연산을 병렬처리 하는 경우와 다수의 개체로 인해 생기는 반복 연산을 병렬처리 하는 경우로 나누어 볼 수 있다. 두 가지 병렬처리에 대한 실험을 해본 결과 고해상도의 영상을 사용할수록 다수의 개체로 인해 생기는 반복연산을 병렬처리를 하는 것이 더 이득이라는 것을 확인할 수 있었다.[13]

Ⅳ. 하드웨어

이번 장에서는 본 논문에서 설계한 하드웨어의 구조 및 수행 시간을 분석한다. 3장에서 설명한 SVM 학습 및 분류를 수행하기 위한 알고리즘을 소개하고 이를 바탕으로 제안하는 하드웨어 구조를 보여준다. 그리고 각 과정의 수행시간을 분석하여 설계한 하드웨어가 임베디드 환경에서도 원활하게 수행할 수 있음을 확인한다.

1. SVM 학습 하드웨어

A. SVM 학습 알고리즘

본 논문에서는 현재 SVM 학습을 위해 가장 많이 사용하고 있는 SMO를 이용하였다. 그림 6은 SMO를 사용하여 SVM 학습을 진행하는 알고리즘의 흐름도이다.[3][6]

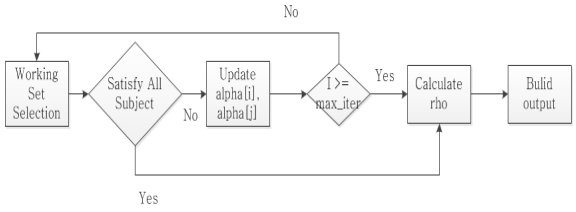

Fig. 6. Flow of SVM Training Algorithm

그림 6. SVM 학습 알고리즘 흐름도

a. Working Set Selection

3장에서 서술한 것과 같이 SVM 학습은 입력 데이터의 수가 많아지면 처리해야 하는 항의 수가 늘어나 계산이 복잡해진다. 이를 해결하기 위해 식 (1)을 계산하는 가장 적은 숫자의 샘플 수인 두 개의 샘플을 고르는 과정을 거치게 되는데 이를 Working Set Selection이라 한다. Working Set Selection을 위해서 식 (2)를 이용한다.

$$\max_{i,j} |E_i - E_j|, E_i \equiv \sum_{i=1}^{l} \alpha_i y_i K(x_i, x_j) - b - y_i \quad (2)$$

이때, E_i는 prediction error로 선택한 i,j를 제외한 나머지를 이용하여 SVM 분류 연산을 수행하였을 때 원래 얻어야 하는 값(positive는 1, negative 는 0)과 차이나는 정도를 의미한다.

b. Update alpha

Working Set Selection을 통해 선택한 두 개의 데이터를 이용하여 식 (1)의 α값을 구해내기 위하여 식 (3),(4),(5)를 계산한다.

$$\alpha_2 = \alpha_2^{old} + \frac{y_2(E_1^{old} - E_2^{old})}{\eta}, \quad (3)$$
$$\eta \equiv K_{11} + K_{22} - 2K_{12}, K_{ij} \equiv K(x_i, x_j)$$

$$\alpha_2^{clipped} = \begin{cases} H & \text{if } \alpha_2 \geq H \\ \alpha_2 & \text{if } L < \alpha_2 < H \\ L & \text{if } \alpha_2 \leq L \end{cases}, \quad (4)$$

$$L = \begin{cases} \max(0, \alpha_2^{old} - \alpha_1^{old}) & \text{if } y_1 \neq y_2 \\ \max(0, \alpha_2^{old} + \alpha_1^{old} + C) & \text{if } y_1 = y_2 \end{cases},$$
$$H = \begin{cases} \min(C, \alpha_2^{old} - \alpha_1^{old} + C) & \text{if } y_1 \neq y_2 \\ \min(C, \alpha_2^{old} + \alpha_1^{old}) & \text{if } y_1 = y_2 \end{cases}$$

$$\alpha_1 = \alpha_1^{old} - s(\alpha_2^{old} - \alpha_2^{clipped}), s \equiv y_1 y_2 \quad (5)$$

여기서 α^{old}는 업데이트하기 전 가지고 있는 값을 의미한다.

c. Calculate b

제한해놓은 반복 횟수를 넘었거나 계산되는 모든 α가 제한조건을 만족하였을 때 식 (6)을 이용하여 b값을 계산한다.

$$b = E_1 + y_1(\alpha_1 - \alpha^{old_1})K(x_1, x_1) \quad (6)$$
$$+ y_2(\alpha_2^{clipped} - \alpha^{old_2})K(x_1, x_2)$$

b값의 계산까지 마무리가 되면 학습된 결과를 파일형태로 출력하여 학습과정을 마무리한다.

B. SVM 학습기 하드웨어 구조

그림 7은 설계한 SVM 학습기의 블록도이다. 학습을 하기 위해서는 입력 데이터의 수가 많기 때문에 FPGA의 내부 SRAM에 전부 저장할 수 없어 DDR 메모리를 이용하였다. 학습할 데이터 전부를 DDR 메모리에 전부 저장한 후 Working Set Selection을 진행한다. Working Set Selection을 진행하면서 계산하는 E_i값은 Cache data memory에 저장되고 그 값을 이용하여 α값을 업데이트 하는데 사용된다. 일련의 연산을 통해 새롭게 바뀐 α값은 다시 DDR 메모리에 저장되어 다음번 반복연산에 다시 사용할 수 있도록 한다.

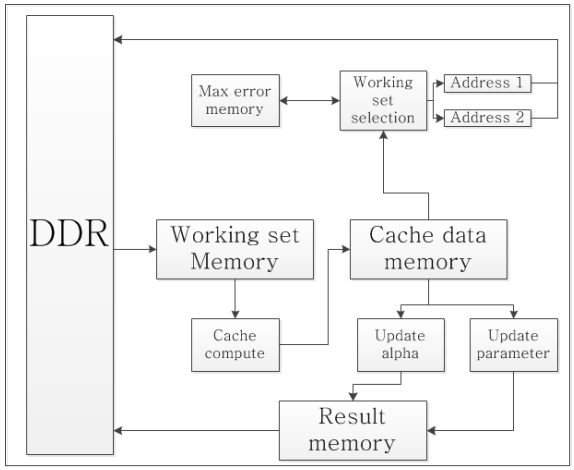

Fig. 7. Block Diagram of SVM Trainer
그림 7. SVM 학습 블록도

그림 6에서 확인 할 수 있듯이 SVM 학습과정은 각 단계별로 상호 의존성을 가지고 있어 앞의 과정이 끝나야지만 다음 과정으로 진행이 가능하다. 따라서 학습의 단계를 병렬화 하여 속도향상을 시키기는 어렵다. 이러한 이유로 설계한 SVM 학습기의 처리속도 향상을 위해서 학습 과정 중 가장 연산부하가 많이 걸리는 부분을 병렬처리하여 속도향상을 시키는 방법을 사용하였다. SVM 학습기에서 가장 연산부하가 많이 걸리는 부분이 E_i값을 계산하는 부분이다. E_i값은 전체 데이터와 커널함수를 계산해야 하는데 이 과정에서 다수의 MAC연산을 실행하게 된다. 이에 따른 연산부하를 해결하기 위하여 MAC 연산을 병렬처리를 하여 수행시간을 단축시켰다. 또한 RBF 커널을 사용함으로써 생기는 지수연산을 테일러 급수를 이용하여 고정소수점 연산이 가능하도록 변경하였다. 변환에 사용한 테일러 급수는 식 (7)과 같다.

$$e^x \fallingdotseq 1 + x + \frac{x^2}{2} \tag{7}$$

C. Timing Analysis

그림 8은 324 차원의 벡터 데이터 2천개를 입력한 후 SVM 학습기가 한번 연산을 수행하는데 소요되는 시간을 분석한 것이다.

Fig. 8. Excution Time Analysis for SVM Trainer
그림 8. SVM 학습 수행시간 분석

Data Input은 학습에 사용할 전체 데이터를 외부버스를 이용하여 DDR 메모리에 저장하는 시간을 나타낸 것이다. DDR 메모리는 FPGA와 AXI 버스를 통해 연결되어 있으며 한 번에 전송되는 데이터의 크기는 32bit 이고 burst length는 192이다. 이렇게 DDR 메모리에 저장된 데이터를 이용하여 학습에 사용할 데이터를 고르는 Working Set Selection을 수행한다. Working Set Selection이 끝난 후 선택된 데이터를 FPGA 내부 메모리로 불러온 후 α및 b값을 업데이트한다.

이러한 과정을 최적의 초평면을 찾아낼 때 까지 반복하게 되는데 2천개의 데이터를 이용하여 실험 하였을 때 총 반복연산은 478번이 수행되었다. 따라서 입력한 데이터를 이용하여 SVM 학습을 하는데 소요되는 시간은 501,439,208 clk이다. 하지만 이 수행시간은 입력한 데이터의 수나 종류에 따라서 변동될 수 있다. 더 자세한 실험 결과는 본 논문의 5장에서 서술한다.

2. SVM 분류 하드웨어

A. SVM 분류 알고리즘

그림 10에 있는 SVM Classifier 모듈은 RBF 커널을 사용한 SVM 분류 결과를 얻기 위해서 식 (8)을 풀어낸다.

$$D(x) = \sum_{i=1}^{SV} (coef * e^{-\gamma^* \| \vec{x} - \vec{v_i} \|^2}) + b \tag{8}$$

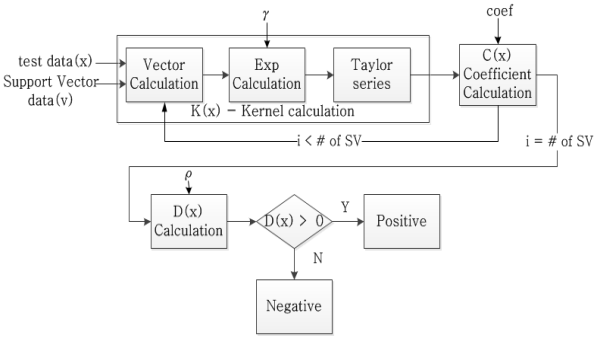

Fig. 9. SVM Classification Flow
그림 9. SVM 분류 흐름도

SVM Classifier 모듈은 Vector Calculation, Exponent Calculation, Taylor Series, Coefficient Calculation 그리고 D(x) Calculation 총 5가지 파트로 나뉜다. 그림 9는 SVM 분류 흐름도이다.

SVM Classifier 모듈의 각 파트별로 수행하는 연산의 내용은 다음과 같다.

a. Vector Calculation

Vector Calculation 파트는 식 (9)를 계산한다.

$$\| \vec{x} - \vec{v} \| = \| \vec{x} \|^2 - 2 * \vec{x} \cdot \vec{v} + \| \vec{v} \|^2 \quad (9)$$

여기서, $\vec{x}$는 테스트 벡터 값이고 $\vec{v}$는 서포트 벡터 값이다. Vector Calculation 파트는 벡터 차원에 따라 최대 350번 반복하여 계산한다.

b. Exponent Calculation

Exponent Calculation 파트는 식 (10)을 계산한다.

$$-\gamma * \| \vec{x} - \vec{v} \|^2 \quad (10)$$

이때, γ는 학습 시 얻어진 데이터 값이다.

c. Taylor Series

식 (11)의 계산을 고정 소수점으로 계산하기 위해 식 (7)과 같이 테일러 급수로 변환하여 계산한다.

$$e^{-\gamma * \| \vec{x} - \vec{v} \|^2} \quad (11)$$

d. Coefficient Calculation

Coefficient Calculation 파트는 식 (12)를 계산한다.

$$coef * e^{-\gamma * \| \vec{x} - \vec{v} \|^2} \quad (12)$$

coef는 학습시 얻어진 데이터 값이다. Coefficient Calculation 파트는 서포트 벡터의 수만큼 최대 500번 반복 계산한다.

e. D(x) Calculation

D(x) Calculation 파트는 전 과정까지 계산된 값을 이용하여 SVM 분류를 위한 식 (8)을 계산하고 최종 결과 값을 만든다. 이때, b는 학습 시 얻어진 데이터 값이다.

최종 결과 값(D(x))을 얻고나서, 그 값이 0보다 크거나 같으면 positive한 샘플이고 0보다 작으면 negative한 샘플로 판단한다. 그리고 positive한 샘플의 경우 flag를 1로하고 negative한 샘플의 경우 flag를 0으로 하여 Output Memory에 저장한다. 저장된 결과는 호스트 PC로 내보내어 질 때 32 bit버스에 8, 16, 24, 32번째 bit에 하나씩 입력하여 내보낸다.

B. SVM 분류기 하드웨어 구조

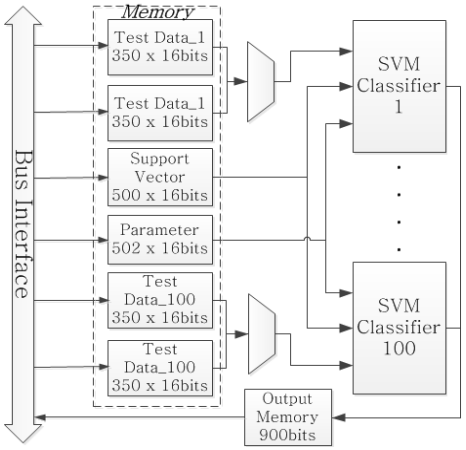

Fig. 10. Block Diagram of SVM Classification Accelerator

그림 10. SVM 분류 가속기 블록도

그림 10은 설계한 SVM 분류 가속기의 블록도이다. 영상의 해상도가 높아지면 입력 영상에서 분류를 하기위한 영역의 수가 늘어나 반복 연산이 늘어나고 전체 수행 시간이 오래 걸리게 되어 실시간 처리에 부적합한 결과가 나타난다. 따라서 본 논문에서는 분류를 할 영역인 마스크(mask) 100개를 동시에 계산함으로써 전체 수행 시간을 줄일 수 있었다. 동시에 계산하는 마스크의 개수를 늘리면 전체 수행 시간이 줄어드는 효과를 얻을 수 있으나 하드웨어 사용량이 늘어나게 되는 단점이 생긴다. 따라서 900개의 마스크를 처리하면서 60 fps (frame/sec) 이상의 성능을 낼 수 있는 최소 조건인 100개의 마스크를 동시에 계산할 수 있도록 설계하였다.

100개의 마스크를 병렬적으로 처리하기 위하여 100개의 SVM 분류기를 배치하고 각각의 분류기에 2개의 테스트 벡터 메모리(Test Vector Memory)를 더블 버퍼의 역할로 연결 하였다. 미리 학습된 서포트 벡터와 RBF 커널을 사용함으로써 학습 시 계산되어 나오는 γ, b, coef와 같은 파라미터를 저장하는 메모리(Support Vectors

Memory, Parameter Memory)는 100개의 분류기가 같은 값을 공유한다.

C. Timing Analysis

그림 11은 설계한 하드웨어가 한 마스크를 수행하는데 소요되는 시간을 분석한 것이다. 그림 11-(a)는 하나의 서포트 벡터에 대한 계산과정을 보여준다. 특징점 추출기의 기술자를 통해 나오는 데이터는 하나의 영상에 대해 하나의 값이 나오는 것이 아니라 다차원의 벡터 형태로 값이 나타난다.

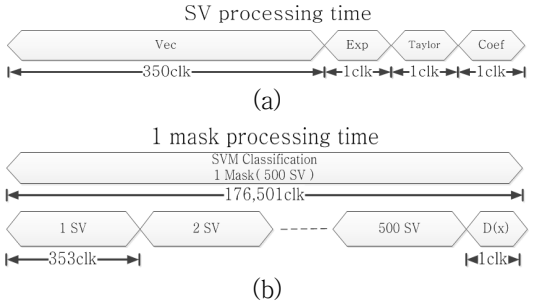

Fig. 11. Excution Time Analysis for 1 Mask
그림 11. 한 마스크에 대한 수행 시간 분석

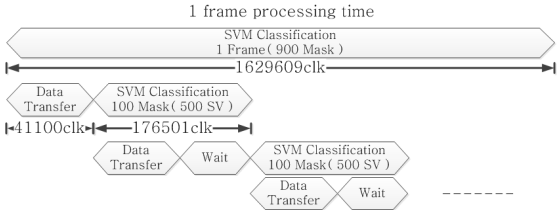

Fig. 12. Excution Time Analysis for 1 Frame
그림 12. 한 프레임에 대한 수행 시간 분석

실제로 학습 시 사용한 특징점 추출기의 종류를 특정하지 않고 다양한 종류의 특징점 추출기와 호환이 가능하도록 서포트 벡터와 테스트 벡터의 최대 차원을 350으로 설계하였다. 또한 학습할 때 사용한 샘플의 수나 학습하는 샘플의 종류에 따라서 나타날 수 있는 서포트 벡터의 수도 정해져 있지 않기 때문에 최대 500개 까지 서포트 벡터를 사용할 수 있도록 설계하였다. 한 마스크에 대한 SVM 분류 결과를 얻기 위해서는 입력 데이터의 차원 수만큼 반복 계산하고 그 값들을 누적시켜 서포트 벡터의 수만큼 반복 연산을 해야 한다. 이러한 내부 반복 연산으로 하나의 마스크에 대해 SVM 분류 결과를 얻기 위해서는 176,501clk이 소요된다. 한 마스크에 대한 수행 시간 분석은 그림 11-(b)를 통해 확인할 수 있다.

그림 12는 한 프레임을 수행하는데 대한 연산 시간을 분석한 것이다. 한 프레임을 수행하기 위해서는 학습을 통해 미리 가지고 있는 서포트 벡터 데이터와 파라미터 데이터를 불러와 저장한다. 하지만 이 과정은 첫 번째 프레임에서 한 번만 진행하면 되기 때문에 이후 프레임에서는 제외된다. 그 후 분류에 사용할 테스트 벡터 데이터를 100개의 마스크 수만큼 불러와 첫 번째 버퍼에 저장한다. 저장이 끝나면 첫 번째 버퍼의 데이터를 이용하여 SVM 분류를 진행하고 동시에 두 번째 버퍼에 다음 100개의 마스크에 대한 테스트 벡터 데이터를 저장한다. 첫 번째 버퍼에 대한 SVM 분류가 끝나면 두 번째 버퍼에 저장된 데이터를 이용하여 SVM 분류를 진행하고 첫 번째 버퍼에 다음 100개의 마스크에 대한 테스트 벡터 데이터를 저장한다. 이와 같은 과정을 반복하여 총 900개의 마스크에 대해 SVM 분류를 진행하고 결과 값을 저장한다. 일련의 과정을 통한 한 프레임의 SVM 분류 결과를 얻기 위해서는 1,629,609clk이 소요된다.

Ⅴ. 실험 및 분석

1. SVM 학습 하드웨어 검증

설계한 하드웨어의 성능을 검증하기 위하여 PC 환경에서 소프트웨어로 수행했을 때와 임베디드 환경에서 소프트웨어만으로 수행했을 때를 비교해 보았다.

Table 2. Comparison of SVM Training Execution Result in Different Experimental Environment
표 2. 실험 환경에 따른 SVM 학습 수행결과 비교

	PC	Em	HW
# of iterations	478	478	478
# of Support Vectors	483	483	483
Execution Time(s)	9.5	35	5

표 2는 실험 환경 변화에 따른 수행결과를 비교한 것이다. SVM 하드웨어는 Verilog HDL을 이용하여 구현하였고 Xilinx Vivado 2014.4.1.을 이

용하여 합성하였으며 동작 주파수는 100MHz이다. 실험에 사용한 PC의 CPU는 Intel i5 3.4GHz이고 RAM은 16GB이며 Windows 7 64bit 운영체제에서 컴파일러는 Visual Studio 2010을 이용하였다. 임베디드 환경은 Xilinx Zynq XC7Z045의 PS(Processing System)로 CPU는 Dual-Core ARM Cortex-A9, 1GB RAM, 운영체제는 Linaro 14.04, 컴파일러는 g++ 4.8.1이다. 실험에 사용한 입력데이터는 TSR 알고리즘에 사용하는 학습 영상 2천장이며 HOG 특징점 추출기를 이용했고 HOG의 크기는 32 x 32이다.

표 2에서 확인할 수 있듯이 PC 및 임베디드 소프트웨어로 수행하였을 때와 하드웨어로 수행하였을 때 반복횟수와 서포트 벡터의 수가 동일함을 보여준다. 이를 통해 본 논문에서 제안하는 하드웨어를 이용하여 SVM 학습을 진행 하였을 때 소프트웨어를 사용하였을 때와 결과가 동일하다는 것을 알 수 있다. 하지만 임베디드 환경에서 소프트웨어만으로 수행 시 소요되는 시간이 하드웨어를 사용하였을 때의 시간과 30초가 차이가 나는 것을 확인할 수 있다. 이를 바탕으로 임베디드 환경에서 소프트웨어만으로 수행했을 때와 하드웨어를 사용하였을 때 어떠한 차이가 나는지 확인해 보았다. 표 3은 임베디드 소프트웨어와 하드웨어의 입력 데이터 수에 따른 수행시간 비교이다.

Table 3. Execution time of SW and HW
표 3. 소프트웨어, 하드웨어 수행시간

# of Samples	# of iterations	SW(ms)	HW(ms)
100	23	1785.16	256.37
500	134	9817.88	1493.62
1000	203	14873.36	2262.73
1500	356	26083.33	3968.13
2000	478	35022.91	5724.74

반복횟수가 늘어감에 따라 소프트웨어와 하드웨어 모두 수행시간이 증가하지만 하드웨어의 증가폭에 비해 소프트웨어의 증가폭이 더 크다는 것을 확인할 수 있다. 이는 많은 데이터를 이용하여 학습을 수행할 경우 소프트웨어만으로는 너무 오랜 시간이 걸려 학습을 효율적으로 수행하기 어려움을 의미한다.

표 4는 제안하는 하드웨어의 성능을 검증하기 위하여 다른 연구와 비교한 것이다. 두 하드웨어의 동작주파수가 달라 단순히 비교하는 것은 어려울 수 있으나 400회 반복 시 [6]의 경우는 425,401,558clk이 소요되는 반면 제안하는 하드웨어는 419,614,400clk이 소요되어 조금 더 빠른시간 안에 처리가 가능함을 알 수 있다.

Table 4. Comparison to Other Research
표 4. 다른 연구와의 비교

# of iterations	[6] (50MHz)	proposed (100MHz)
200	4.85s	2.11s
300	6.59s	3.17s
400	8.51s	4.22s

2. SVM 분류 하드웨어 검증

제안하는 하드웨어의 성능을 확인하기 위하여 TSR 알고리즘에 적용하였다. TSR 알고리즘에 적용하여 SVM 분류기만을 검증하는 것이 아닌 하드웨어를 이용하여 학습한 데이터들이 소프트웨어로 학습한 데이터와 비교하였을 때 차이가 없음을 같이 검증하였다. 그림 13은 Xilinx ZC706 보드에 구현한 실험 환경의 블록도이다.

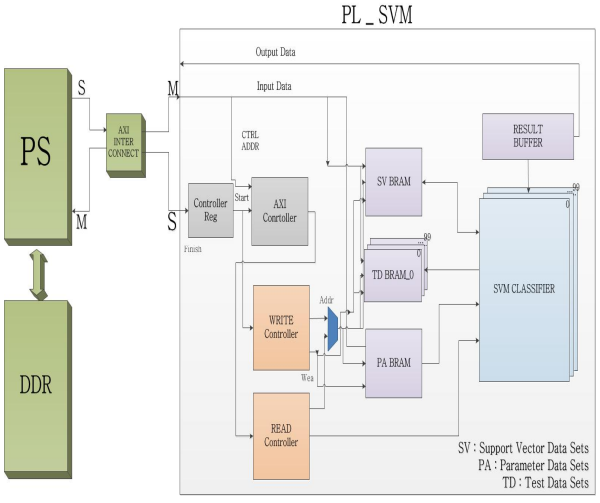

Fig. 13. Block Diagram for Experimental Environment

그림 13. 실험환경 블록도

실험환경의 호스트 PC 및 소프트웨어 부분은 학습 부분 검증에서 사용한 임베디드 환경이며 영상 입출력을 위해 OpenCV 2.4.9 라이브러리를 사용하였다. 하드웨어를 구현하는데 사용한 언어, 합성 툴, 그리고 동작주파수도 학습 과정과 동일하다. SVM 분류기의 성능을 확인하기 위해 사용

한 영상은 1360 x 800 해상도의 독일 벤치마크 영상[19]이며, 표지판 영역을 판단할 마스크의 크기는 40 x 40이고, 사용한 특징점 추출기는 HOG이다. 학습 및 분류에 사용한 HOG의 크기는 32 x 32이다.

그림 14는 실험에 사용한 TSR 알고리즘의 순서도이다. TSR 알고리즘은 입력영상에서 표지판이 있을 법한 영역을 분할(segmentation)한 후 분할한 영역을 마스크를 이동하며 HOG 특징점 추출기를 이용하여 특징점을 뽑아낸다. 그리고 추출한 특징점을 학습 시 사용한 매핑 값을 기준으로 -1부터 1사이로 매핑하고 분류를 실시하고 SVM 분류로 나온 결과가 0 이상이면 positive로 판단하고 마스크 크기만큼 표지판 영역임을 표시한다. SVM 분류 후 마지막 영상이면 종료하고 아니면 처음부터 다시 수행한다.

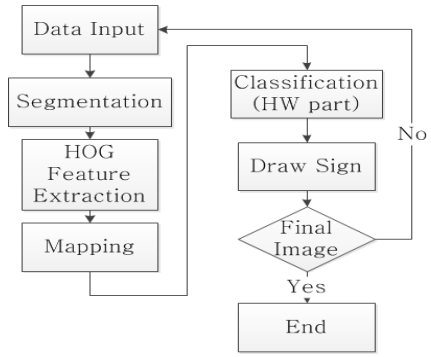

Fig. 14. TSR Algorithm Flow
그림 14. TSR 알고리즘 순서도

TSR 알고리즘 중에서 본 논문에서 제안하는 하드웨어 가속기를 사용하는 부분은 매핑 이후 분류를 하는 부분이다. 하드웨어 부분을 제외한 나머지는 ZC706 board의 PS에서 처리하여 그림 15과 같이 정상적으로 표지판을 표시하는 것을 확인하였다.

Fig. 15. TSR Result Image
그림 15. TSR 결과영상

Table 5. Comparison of TSR Result
표 5. TSR 수행결과 비교

	All SW	SW	HW
# of traffic signs	986	986	986
# of detected signs	943	943	943
Detection rate (%)	95.6	95.6	95.6

표 5는 TSR 알고리즘 수행 시 검출률을 비교한 것이다. 소프트웨어로 학습한 데이터를 이용하여 소프트웨어로 분류를 진행한 경우(All SW)와 하드웨어로 학습하고 소프트웨어로 분류를 진행한 경우(SW), 그리고 하드웨어로 학습하고 하드웨어로 분류를 진행한 경우(HW)로 나누어 실험을 진행하였다. 독일 벤치마크 영상 900장을 이용하여 실험을 진행하였고 검출률을 구하는 방법은 식 (13)와 같다.

$$Detection\ rate = \frac{\#\ of\ detected\ signs}{\#\ of\ all\ signs} \quad (13)$$

TSR 알고리즘을 이용한 실험 결과 소프트웨어만으로 수행하였을 때와 설계한 하드웨어 가속기를 이용하였을 때 같은 결과를 보였다. 실험을 진행한 벤치마크 영상 900장에 총 986개의 표지판이 있는데 세 가지 경우 모두 그 중 943개를 검출하여 95.6%의 검출률을 보였다. 이를 통해 본 논문에서 제안하는 고정 소수점을 사용한 학습 및 분류 하드웨어의 성능이 부동 소수점을 사용하는 소프트웨어의 성능과 비슷하다는 것을 확인할 수 있다.

Table 6. Execution Time of SW and HW
표 6. 소프트웨어, 하드웨어 수행 시간

	SW (ms)	SW + HW (ms)
Data Input	321.14	321.14
Segmentation	3378.35	3378.35
HOG	491.88	491.88
Mapping	45.76	45.76
Classification	**422000**	**21.34 (HW part)**
Output	18.11	18.11
Total	426255.24	4280.34

표 6은 실험환경에서 TSR 알고리즘을 소프트웨어만으로 수행했을 때와 본 논문에서 제안하는 분류 하드웨어 가속기를 이용하여 수행했을 때의 수행 시간을 분석해 놓은 것이다. Classification 부분이 소프트웨어만을 이용했을 때는

422,000ms(422sec)가 소요되지만 하드웨어 가속기를 이용하면 21.34ms가 소요됨을 확인할 수 있다. 하지만 이 소요시간에는 하드웨어가 한 프레임 데이터를 처리하는 동안 소프트웨어는 다음 프레임 데이터를 준비하는 시간이 포함되어있다. 이는 실제 시스템의 소요시간에는 영향을 미치지 않는 시간이다. 이를 분석하기 위한 하드웨어 구간별 수행 시간은 표 7과 같다. 표 7에서 MMAP은 소프트웨어에서 하드웨어로 보내기 위해 PS DDR 메모리에 맵핑하는 과정으로 하드웨어가 연산을 수행하는 동안 소프트웨어에서 미리 처리하여 소모되지 않는 시간이다. Data Write는 하드웨어에서 사용할 데이터를 FPGA의 BRAM(Block RAM)에 저장하는 부분이고 Data Read는 SVM 분류 결과를 소프트웨어가 읽어오는 시간이다. 따라서 실제적으로 하드웨어 수행 시간은 16.54ms (약 60.46 fps)이다. 이를 통해 제안하는 하드웨어가 실시간 처리가 가능함을 확인할 수 있다.

Table 7. Execution Time for Each HW Section
표 7. 하드웨어 구간별 수행 시간

	Excution time for one frame(ms)
MMAP	4.8
Data Write	0.4
SVM Classification	15.94
Data Read	0.2
Total	21.34

Table 8. Performance Comparison with Other Classifier
표 8. 타 논문의 분류기와 성능비교

	[7]	[12]	[20]	proposed
Device	Vertex5	GPU	Vertex6	Zynq
Operating frequency	50MHz	63MHz	40MHz	100MHz
Used feature point extractor	SIFT	HOG	LBP	any kind of descriptor
# of input dimensions	500	1980	1440	350
# of SVs	100	-	-	500
Parallel architecture (# of unit)	SVU (20)	SVU (-)	SVU (-)	CL (100)
Processing speed (# of masks)	4.44fps (900)	11fps (900)	60fps (19200)	60.46fps (900)
Used kernel	RBF	RBF	Linear	RBF

본 논문에서 설계한 하드웨어와 동일한 구조를 가지는 논문을 찾을 수 없어 제안하는 하드웨어와 유사한 구조를 가지는 논문들과 성능을 비교하였다. 표 8은 본 논문과 타 논문과의 성능비교를 보여준다. 여기서 SVU는 다수의 서포트 벡터로 인해 생기는 반복연산을 병렬처리 한 것을 말하고 CL은 SVM 분류기를 병렬처리하여 다수의 개체를 동시에 처리하도록 한 것을 의미한다.[13]

SVU 병렬처리 하여 SVM 분류를 가속시킨 [7]의 하드웨어는 사용하는 특징점 추출기를 SIFT으로 고정시키고 입력 벡터는 최대 500차원이며 서포트 벡터의 수는 최대 100이다. 속도향상을 위한 병렬구조는 20개의 SVU를 이용하였고 900개의 후보군을 처리하는데 225ms가 소요되어 실시간 처리가 제한된다. GPU를 이용하여 SVM 분류를 가속한 [12]는 900개의 마스크를 처리하는데 90ms가 걸려 실시간 처리에 한계가 있을 뿐 아니라 GPU의 특성상 임베디드 환경에서 전력 소모가 많은 한계점이 있다. 선형 SVM을 사용한 [19]는 LBP(Local Binary Pattern) 특징점을 이용한 머리-어깨 탐지 시스템으로 640 X 480 해상도에서 19200개의 후보군에 대해 60fps로 동작한다. 이는 선형 SVM의 특성상 분류속도가 빠르기 때문에 가능하지만 선형 SVM을 사용하기 때문에 커널 기법을 사용한 것에 비해 낮은 SVM 분류 성능을 가진다. 따라서 SVM의 성능을 보완하기 위해 추가적으로 전경검출 알고리즘까지 추가하는 단점이 있다. 이들에 비해 제안하는 하드웨어는 특징점 추출기를 고정하지 않고 다양한 특징점 추출기를 이용할 수 있고 입력 벡터는 최대 350차원, 서포트 벡터는 최대 500개이다.

본 논문에서는 처리 속도 향상을 위해 동시에 100개의 마스크를 계산하여 수행 시간을 줄이는 반면 [7]에서는 내부 반복연산 모듈을 동시에 20개를 수행하는 차이가 있다. 이로 인해 100개의 마스크를 처리하는 수행 시간은 [7]의 경우 25ms이지만 제안하는 하드웨어는 1.77ms이다. 또한 900개의 마스크에 대해 [7]의 구조는 225ms가 소요되어 실시간 처리가 불가능하지만 제안한 구조는 16.54ms로 실시간 처리가 가능하다. 이를 통해 본 논문에서 제안하는 하드웨어가 실시간 처리를 요하는 고속 시스템에 사용하기 더 적합함을 확인할 수 있다.

Ⅵ. 결론

본 논문에서는 영상처리 분야에서 많이 사용하는 기계학습 방법인 SVM 알고리즘을 하드웨어로 구현하여 임베디드 환경에서 SVM 학습 및 분류가 가능한 통합 시스템을 제안하였다. SVM 학습을 위해서 SMO 방식을 기반으로 하였고 학습의 정확도를 높이기 위해 RBF 커널을 이용하였다. 학습을 위한 연산과정 중 반복 연산으로 인해 수행 시간의 저하를 유발하는 E_i의 연산을 병렬처리를 통하여 수행 시간을 단축시키고 E_i의 cache 메모리를 이용하여 계산한 값의 반복사용을 통해 수행 시간을 단축할 수 있었다. 또한 커널사용으로 인해 생기는 지수연산을 테일러 급수를 이용하여 고정 소수점 연산이 가능하도록 하드웨어를 설계하였다. 그리고 SVM 분류기는 영상의 해상도가 높아짐에 따라 나타나는 반복 연산으로 인한 수행 시간의 저하를 줄이기 위하여 여러 개의 분류기를 병렬로 설계하여 수행 시간을 단축하였다. SVM 분류에도 학습과 동일하게 RBF 커널을 사용하였기 때문에 테일러 급수를 이용하여 고정 소수점 연산이 가능하도록 하드웨어를 설계하였다. 설계한 SVM 통합 시스템의 검증을 위하여 TSR 알고리즘을 이용하였는데 하드웨어를 사용하여 학습 시 동작 주파수 100MHz에서 5초의 시간이 소요되어 PC에서 학습을 하는데 걸리는 시간인 9.5초 보다 빠르게 수행하는 것을 확인할 수 있었다. 그리고 하드웨어로 학습한 결과와 소프트웨어로 학습한 결과를 TSR 알고리즘에 적용하여 동일한 검출률인 95.6%를 나타내는 것을 확인하여 하드웨어의 고정 소수점을 이용한 연산이 소프트웨어의 부동 소수점 연산의 결과와 차이가 없음을 검증하였다. 그리고 설계한 하드웨어를 이용하여 TSR 알고리즘을 수행하였을 때 동작 주파수 100MHz에서 16.54ms가 소요되어 실시간 처리가 가능함을 확인하였다. 따라서 본 논문에서 제안하는 SVM 통합 시스템이 임베디드 환경에서 학습과정 및 분류과정 모두 원할히 수행이 가능함을 확인했다.

향후 과제는 현재 분류기와 학습기가 따로 구현이 되어 하드웨어 리소스를 공유하지 못하는데 SVM 학습 및 분류에서 반복 연산을 병렬처리 하는데 핵심이 되는 연산인 커널함수 계산하는 부분을 공유함으로써 하드웨어 리소스를 감소시켜 임베디드 환경에 더욱 적합한 하드웨어를 개발하는 것이다.

References

[1] K. Grauman and T. Darrell, “The pyramid match kernel Discriminative classification with sets of image features”, *Proceedings of IEEE International Conference on Computer Vision*, 2005

[2] Y. H. Kang, Y. B. Park, “Design of Automatic Document Classifier for IT documents based on SVM,
” *j.inst.Korean.electr.electron.eng*, Vol. 8, No. 2 186-194, 2004

[3] J. Platt, “Fast Training of Support Vector Machines Using Sequential Minimal Optimization,” in *Advances in kernel methods - Support Vector Learning*, MIT Press, 185-208, 1999

[4] C. Chang and C. Lin, “LIBSVM : A Library for Support Vector Machine,” *ACM Transactions on Intelligent Systems and Technology*, Vol 2, No.3, 1-27, 2011

[5] S. Cadambi, I. Srihari, et al. "A massively parallel FPGA-based coprocessor for support vector machines," *Field Programmable Custom Computing Machines '09. 17th IEEE Symposium on*, Napa, USA, 2009

[6] T. Kuan, J. Wang, J. Wang, P. Lin, G. Gu, “VLSI Design of an SVM Learning Core on Sequential Minimal Optimization Algorithm,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 20, No. 4, 673-683, 2012

[7] M. Qasaimeh, A. Sagahyroon, and T. Shanableh, “FPGA-Based Parallel Hardware Architecture for Real-Time Image Classification,” *Computational Imaging, IEEE Transactions*, Vol. 1, No. 1 56-70, 2015

[8] Marta Ruiz-Llata, Guillermo Guarnizo and Mar Yébenes-Calvino, “FPGA Implementation of a Support Vector Machine for Classification and Regression,” *IJCNN*, 2010

[9] Xipeng Pan, Huihua Yang, Lingqiao Li, Zhenbing Liu, Le Hou , "FPGA Implementation of SVM Decision Function Based on Hardware-friendly Kernel," *ICCIS*, 2013

[10] Bryan Catanzaro, “Fast Support Vector Machine Training and Classification on Graphics Processors,” *IMAGING*, Vol. 1, No. 1, March 2015

[11] Christos Kyrkou, Theocharis Theocharides, "A Parallel Hardware Architecture for Real-Time Object Detection with Support Vector Machines," *IEEE TRANSACTIONS ON COMPUTERS*, Vol. 61, No. 6, 2012

[12] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, “A Practical Guide to Support Vector Classification,” *Technical Report*, Department of Computer Science National Taiwan University, 2003

[13] W. S. Na, S. W. Han, Y. J. Jeong, “FPGA Design of SVM Classifier for Real Time Image Processing,” *j.inst.Korean.electr.electron.eng*, Vol. 20, No. 3 209-219, 2016

[14] C. Cortes and V. Vapnik, “Support-Vector Network,” *Machine Learning*, Vol. 20, No. 3 273-297, 1995

[15] Il-Seok Oh, *Pattern Recognition*, Kyobomoongo, 137-173, 2008

[16] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*, 2008

[17] V. Vapnik, Estimation of Dependances Based on Empirical Data, Springer-Verlag, 1982

[18] E. Osuna, R. Freund, F. Girosi, “An improved training algorithm for support vector machines,” *Neural Networks for Signal Processing VII. Proceedings of the 1997 IEEE Workshop*, 276-285, 1997

[19] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition,” *Neural Networks*, vol. 32, pp. 323 - 332, 2012.

[20] Tomasz Kryjak, Mateusz Komorkiewicz, Marek Gorgon, “FPGA IMPLEMENTATION OF REAL-TIME HEAD-SHOULDER DETECTION USING LOCAL BINARY PATTERNS, SVM AND FOREGROUND OBJECT DETECTION,” *Design and Architectures for Signal and Image Processing (DASIP) Conference*, Karlsruhe, Germany, 2012

BIOGRAPHY

Wonseob Na (Student Member)

2014 : BS degree in Electronics and Communications Engineering, Kwangwoon University.
2014~ : Course of MS in Electronics and Communications Engineering, Kwangwoon University.

Yongjin Jeong (Member)

1983 : BS degree in Control and Instrumentation Engineering, Seoul National University.
1995 : MS, PhD degree in Electronics and Computer Engineering, University of Massachusetts, Amherst
1983~1989 : Research Engineer, ETRI
1995~1999 : Chief Researcher, Samsung Electronics
1999~current : Professor, Dept. of Electronics and Communications Engineering, Kwangwoon Univ.