한국정보전자통신기술학회논문지(jkiiect)'19-2, Vol.12 No.1
논문 19-12-01-29 http://dx.doi.org/10.17661/jkiiect.2019.12.1.29

추가 학습이 빈번히 필요한 비포장도로에서 주행로 탐색에 적합한 GLSL 기반 ALNN Algorithm

구본우* 김준겸* 이은주**

GLSL based Additional Learning Nearest Neighbor Algorithm suitable for Locating Unpaved Road

Bon Woo Ku*, Jun kyum Kim* and Eun Joo Rhee**

요 약 국방 분야에서 무인 차량의 주행로는 포장 도로 뿐만 아니라, 자주 다양한 변화를 갖는 야지의 비포장 도로 등이 포함된다. 이 무인 차량은 주로 험지나 오지에서 감시 및 정찰, 진지 방어 등을 수행하므로 자율 주행을 위해서 예측하지 못했던 다양한 주행로와 환경을 수시로 접하게 되며, 이에 따라 추가 학습이 필요하다. 본 논문에서는 'Forgetting' 문제를 피하면서 거리 비교와 Class 비교를 통해 빠르게 추가 학습이 가능하도록 Approximate Nearest Neighbor를 수정한 GPU 기반 Additional Learning Nearest Neighbor(ALNN) 알고리즘을 제안한다. 또 ALNN 알고리즘은 학습 데이터가 누적될수록 연산 속도가 저하되는 문제가 있고, 본 연구에서는 OpenGL Shading Language 기반의 GPU 병렬 처리를 사용하여 이를 해결하였다. ALNN 알고리즘은 기존의 학습 데이터에 영향을 주지 않으면서 빠르게 추가 학습이 가능하여, 빈번히 실시간으로 재학습이 필요한 국방 등의 분야에 활용될 수 있다.

Abstract Unmanned Autonomous Vehicle's driving road in the national defense includes not only paved roads, but also unpaved roads which have rough and unexpected changes. This Unmanned Autonomous Vehicles monitor and recon rugged or remote areas, and defend own position, they frequently encounter environments roads of various and unpredictable. Thus, they need additional learning to drive in this environment, we propose a Additional Learning Nearest Neighbor (ALNN) which is modified from Approximate Nearest Neighbor to allow for quick learning while avoiding the 'Forgetting' problem. In addition, since the Execution speed of the ALNN algorithm decreases as the learning data accumulates, we also propose a solution to this problem using GPU parallel processing based on OpenGL Shader Language. The ALNN based on GPU algorithm can be used in the field of national defense and other similar fields, which require frequent and quick application of additional learning in real-time without affecting the existing learning data.

Key Words : Approximate Nearest Neighbor, Additional Learning, GPU Parallel Processing, Unmanned Autonomous Vehicle

1. 서론

국방 분야에서의 무인 차량은 인간 전투원이 투입되기 어려운 열악한 환경에서 개략적인 사전 정보와 탑재된 센서로 부터 얻는 정보를 활용하여 자율주행 임무를 수행한다. 자율 주행 임무가 부여된 무인 차량의 주행로는 자갈, 바위, 초목 등으로 이루어져 있으며 노면의 굴곡, 지표 특성 등 환경이 쉽게 변하고, 주행로 정보가 포함된 정밀한 지도가 없는 야지 환경[1]의 주행로가 주를 이룬다. 이에 따라 주행로

*M&S 1 Department, Simnet Co., Ltd.
**Corresponding Author : Department of Computer Engineering, Hanbat National University (ejrhee@hanbat.ac.kr)
Received January 22, 2019 Revised February 01, 2019 Accepted February 11, 2019

를 검출하고 새롭게 변화된 환경에 적응하기 위해서는 추가 학습이 가능한 분류기가 필요하다.

그간 컴퓨터 비전 분야에서 자율 주행 차량의 주행 영역 검출은 카메라와 기계 학습 기술을 활용하여 다양하게 연구를 수행하였다. Shang, Erke 등은 KNN, SVM과 Confidence Map을 결합하여 만든 알고리즘으로 비포장도로를 검출하였다[2]. Abhijit Kundu 등은 영상의 시간 정보와 특징 벡터 최적화를 통해 CRF로 연속적으로 변하는 물체를 인식하는데 성공하였다[3].

그러나 이러한 방법들은 추가 학습을 수행할 수 없는 구조로, 빈번히 변화되는 환경에서 운용되는 국방 분야의 무인 차량에 적용하기에는 부적합하다. 왜냐하면 이 방법들은 추가 학습할 때 기존의 학습 데이터가 손실되는 'Forgetting' 문제가 발생되기 때문이다[4]. 따라서 Martha Roseberry 등은 KNN에 Self-Adjusting Memory(SAM)를 적용하여 'Forgetting' 문제를 최소화하려 하였지만, 완전한 답을 얻지 못했다[5].

본 논문에서는 국방 분야 등의 무인 차량 자율주행에서 가장 큰 문제인 'Forgetting' 문제가 발생하지 않으면서 추가 학습이 가능하도록, ANN (Approximate Nearest Neighbor)을 수정•보완한 ALNN(Additional Learning Nearest Neighbor) 알고리즘을 제안한다. 제안한 ALNN 알고리즘은 학습 데이터가 누적될수록 연산 속도가 저하되는 문제가 발생하며, GLSL 기반의 GPU 병렬 처리를 사용하여 이를 해결하고자 한다.

2. 특징 벡터 추출

ALNN 알고리즘은 이미지에서 추출한 특징 벡터를 이용하여 이미지를 학습하고 분류한다. 본 연구에서본 구제안한 특징 벡터는 이미지에서 일정 범위 내 픽셀들의 관계 정보, 각 픽셀의 R, G, B의 색상 정보와 HOG(Histogram of oriented gradients)[6]인 윤곽선 분포 정보로 구성되며, 그림 1.과 같다. HOG는 픽셀들의 Edge의 양과 방향을 나타내며, 회전에 영향을 받지 않는 다는 장점이 있다.

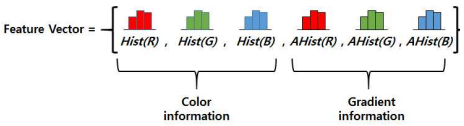

그림 1. 특징 벡터.
Fig. 1. Feature Vector.

$$gradient = \sqrt{dX^2 + dY^2} \tag{1}$$

식(1)은 그림 1.의 HOG에 적용되는 $gradient$의 계산식이다. 여기서, $gradient$는 X축과 Y축의 미분값인 dX,dY에 의해 구하고, HOG는 각 화소의 R, G, B Histogram에 의하여 구한다. 식(1)의 dX,dY는 식(2)와 같이 Sobel 미분 Mask를 Img의 모든 픽셀에 적용한 것이다. 여기서 Img는 입력된 이미지이고, $(-1,0,1)$은 Sobel Mask의 X축 미분 Mask를 지칭한다. 또 $(-1,0,1)^T$는 Sobel Mask의 Y축 미분 Mask이다.

$$\begin{aligned} dX &= Img \otimes (-1,0,1) = \partial Img/\partial x \\ dY &= Img \otimes (-1,0,1)^T = \partial Img/\partial y \end{aligned} \tag{2}$$

3. ALNN 알고리즘

ALNN은 ANN을 기반으로 'Forgetting' 문제가 발생하지 않도록 개선한 것이다. ANN 알고리즘은 그림 2.와 같이 특정점에서 거리를 계산하여 일정 거리내의 데이터들을 그룹화 하여 데이터 분류를 한다[7].

ANN 알고리즘은 모든 입력 데이터들의 중요도를 동일시한다. 어떤 데이터가 노이즈(Noise)이고 어떤 데이터가 우수한 데이터인지 판별하지 않으며, 오로지 과반수 의결에 의해 Class를 정의한다. 그래서 ANN 알고리즘은 추가 학습을 수행할 때, 기존의 학습 데이터와 추세가 유사한 학습 데이터가 입력되면 문제가 없다. 반면에 추세가 유사하지 않은 학습 데이터가 입력되면 기존의 학습 데이터가 노이즈로 변하는 'Forgetting' 문제가 발생한다.

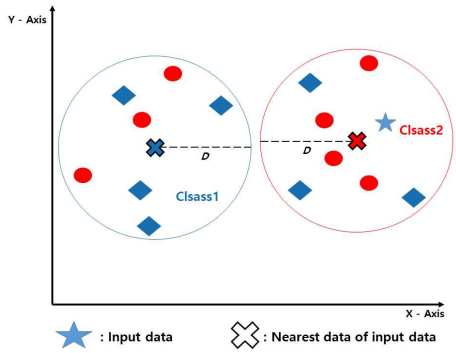

그림 2. ANN 알고리즘.
Fig. 2. ANN Algorithm.

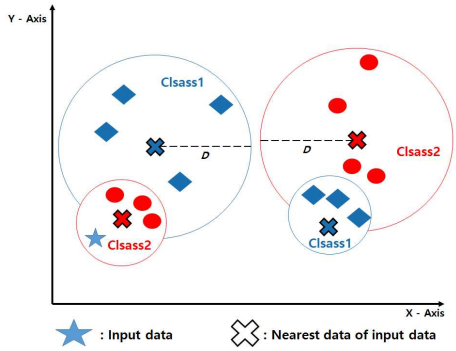

그림 3. ALNN 알고리즘.
Fig. 3. ALNN Algorithm.

제안한 ALNN 알고리즘은 ANN에서 사용하는 유클리드 거리(Euclidean distance)에 의한 최근접 데이터 선택 기법을 그대로 채용한다. 단 그림 3.에서와 같이 소수 데이터의 Class를 별도로 구분하여 학습 데이터에 추하는 차이가 있다. 따라서 이 방법은 소수 데이터의 Class가 소실되지 않고, 학습 데이터에 보존되기 때문에 추가 학습에서 'Forgetting' 문제가 발생하지 않게 된다.

식(3)은 본 방법에서 사용한 유클리드 거리에 의한 최근접 데이터 선택 기법을 나타낸 것이다. 여기서 $\dot{id}$는 입력 데이터와 학습 데이터의 거리를 계산에 의해 선택된 최근접 데이터를 지칭한다.

$$\dot{id} = \mathrm{arg}min_{id} dist(InputData, TrainData) \quad (3)$$

학습에서 학습 데이터 추가는, 입력 데이터와 학습 데이터 사이의 거리 비교와 Class 비교를 수행하고, 그 결과에 따라 학습 데이터에 추가되거나 추가되지 않게 된다. 식(4)와 식(5)에서 이를 보인다.

식(4)에서 D는 데이터를 그룹화를 위한 거리이며, 실험에서 얻은 최적값이다. I^F는 입력 데이터의 집합이다. T^F는 ALNN 알고리즘으로 학습한 데이터들의 집합이고, i는 식(3)의 $\dot{id}$를 의미한다. $DF(i,j)$는 I_j^F와 T_i^F의 거리를 계산하고, D와 비교하는 함수이다. 만약 두 데이터의 거리가 D보다 적다면 같은 이웃 데이터로 보고 식 (5)의 $MF(i,j)$함수를 수행한다. 반대로 두 데이터의 거리가 D보다 크다면 새로운 경우로 판별하고 $AddData(I_j^F)$함수를 통해 T^F에 I_j^F를 추가한다.

$$DF(i,j) = \begin{cases} MF(i,j); & \text{if } dist(I_i^F, T_j^F) \le D \\ AddData(I_i^F); & else \end{cases} \quad (4)$$
$$, j \in [1 \sim I^F.Size]$$

$MF(i,j)$함수는 I_j^F와 T_i^F의 Class를 비교하여, Class가 동일하지 않으면 새로운 Class로 판별하고 $AddData(I_j^F)$함수를 통해 T^F에 I_j^F를 추가한다. 반대로 Class가 동일하면, 입력 데이터를 다음 데이터로 변경한다.

$$MF(i,j) = \begin{cases} AddData(I_i^F); & \text{if } I_i^F.Class \neq T_j^F.Class \\ ; & else \end{cases} \quad (5)$$

$$DF(\dot{id}, j) \quad (6)$$
$$, \dot{id} = \mathrm{arg}min_{id} dist(I_j^F, T_{id}^F)$$
$$, j \in [1 \sim I^F.Size]$$

식(6)은 ALNN 알고리즘을 수식으로 나타낸 것으로 식(3)과 (5)를 통합한 것이다. 여기서 I_j^F와 T_{id}^F는 입력 데이터와 학습 데이터를 나타낸다. $\mathrm{arg}min_{id}$함수는 I_j^F를 기준으로 T_{id}^F에서 최근접 학습 데이터를 선택하여 $\dot{id}$에 할당한다.

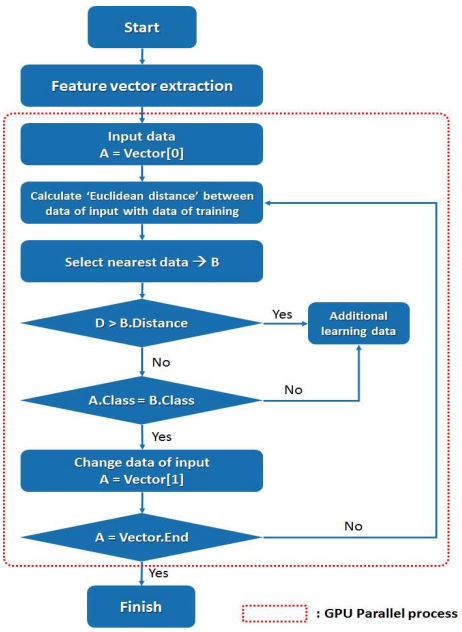

그림 4. ALNN 알고리즘의 절차.
Fig. 4. Procedure of ALNN Algorithm.

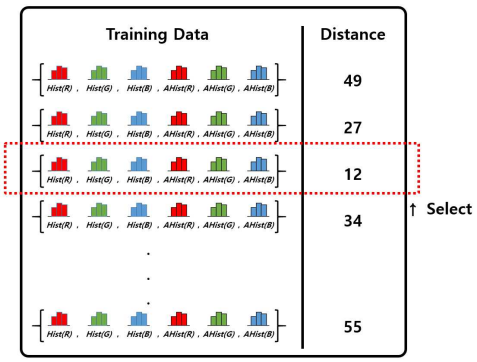

그림 5. 최근접 데이터의 예.
Fig. 5. Example of nearest data.

ALNN 알고리즘에서 학습은 식(4)와 (5)에 의해 입력 데이터를 기존의 학습 데이터와 비교한 후 신규 데이터를 추가하므로 기존의 학습데이터 손실 없이 보존된다. ALNN의 실행 과정은 그림 4.에, 학습 과정에서 새로운 입력 데이터와 기존 학습 데이터와의 거리 예를 그림 5.에 보인다..

3.1 GPU Parallel Process Using GLSL

ALNN 알고리즘은 구조상 학습이 지속적으로 수행될 때마다 학습 데이터의 크기가 빠르게 증가한다. 이에 입력 데이터와 ALNN의 학습 데이터 간의 거리를 계산하고 비교하는 연산 속도가 지속적으로 저하된다. 본 논문에서는 이를 해결하기 위해 GLSL(OpenGL Shading Language) 기반의 GPU 병렬 처리를 ALNN 알고리즘에 적용하였다.

GLSL은 Khronos Group에 의해 개발된 그래픽 라이브러리이며[8], 게임과 시각화 등 3D 그래픽을 분야에 널리 사용되고 있다. GLSL은 Vertex shader와 Fragment shader라는 두 가지 연산 방식이 있다. Vertex Shader는 입력된 각 Vertex의 위치를 GPU의 각 코어 당 하나씩 맡아 병렬로 처리하는 Shader이고, Fragment shader는 정해진 Vertex에 의해 생성된 Mesh의 색상을 연산하는 Shader이다[9].

제안한 ALNN의 연산에서는 GPU의 각 코어를 하나씩 맡아 병렬로 처리하는 Vertex Shader를 사용하였다. 그림 6.은 ALNN 알고리즘에 GLSL 기반 GPU 병렬 처리 적용 방법을 도식화한 것이며, 실행 과정은 아래의 [GPU에 의한 병렬 처리]와 같다.

[GPU에 의한 병렬 처리]

① 입력된 이미지에서 특징 벡터를 추출한다. ALNN은 순차적으로 추출된 이 특징 벡터들을 입력 데이터로 사용한다.

② 입력 데이터는 Vertex Shader를 사용하여 Vertex형태로 바꾸어 GPU의 각 코어에 병렬로 할당한다.

③ ALNN의 학습 데이터를 3D Texture(OpenGL 지원)를 사용하여, 모든 학습 데이터를 GPU 메모리에 로드한다.

④ 그림 4.의 ALNN 수행 절차에서 'GPU 병렬처리' 부분을 각 코어에서 병렬로 처리한다.

⑤ 모든 입력 데이터에 대하여 과정 ①~④를 반복 실행하면 분류 결과 이미지를 생성되고 절차는 종료된다.

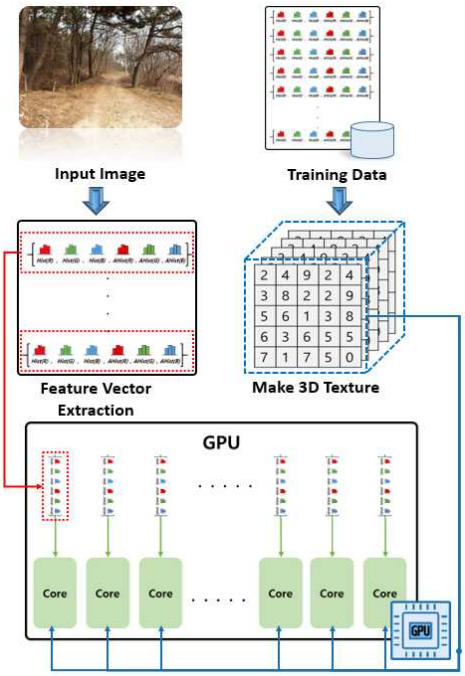

그림 6. GLSL을 사용한 GPU 병렬 처리.
Fig. 6. GPU Parallel Process using GLSL

4. 실험 및 고찰

본 연구에서, 실험은 i7-7700 CPU와 NVDIA Geforce GTX 1080Ti(11GB) 그래픽카드를 탑재한 PC를 사용하였다. 사용한 언어 및 라이브러리는 C/C++, OpenGL 4.0과 OpenCV 3.0이다. 실험에 사용한 이미지는 비포장도로를 촬영한 181장의 이미지와 이것의 GroundTruth이다. 이이미지는 training에 100장과 test에 80장으로 분류하였고, training 이미지는 Data augmentation에 의하여 총 1000장으로 증가시켰다. 본 연구에서는 제안한 알고리즘의 유용성 검토를 위하여 3종의 실험을 하였다.

실험 1과 2는 제안한 ALNN 알고리즘에서 'Forgetting' 문제가 발생되지 않음을 확인하기 위한 것이다. 실험 1은 ALNN 알고리즘에 대한 추가 학습 실험으로, 총 10회의 추가 학습을 수행하고 매 회마다 분류 결과 이미지에 대한 픽셀 정확도를 측정하였다. 실험 2는 ALNN 알고리즘의 대조군으로 택한 ANN 알고리즘에 대한 추가 학습 실험으로, 실험 1과 동일한 방법으로 수행되었다.

실험 3은 ALNN 알고리즘의 연산 속도 측정 실험이다. ALNN 알고리즘을 GLSL 기반 GPU 병렬처리를 적용하였을 때와 적용하지 않았을 때의 연산 속도를 측정하고 이를 비교하여 ALNN 알고리즘의 활용 가능성을 고찰하였다.

4.1 ALNN 알고리즘 추가 학습 실험

실험 1은 ALNN 알고리즘의 추가 학습에서 'Forgetting' 문제가 발생하지 않음을 증명하기 위한 것이다. 이 실험에서는 추가 학습을 수행하고 분류 결과 이미지의 픽셀 정확도를 측정한다.

$$MatchingRate(\%) = \frac{CP^{nm}}{GP^{nm}} \times 100 \qquad (7)$$

식(7)은 픽셀 정확도 계산을 위한 식이다. 여기서 GP^{nm}는 GroundTruth의 픽셀수, CP^{nm}은 분류 결과 이미지에서 GroundTruth와 동일한 픽셀수 즉 정확하게 분류된 픽셀의 수이다.

실험에서는 ALNN에 의하여 training 이미지 100장을 학습하여 학습 데이터를 생성한 후, test 이미지 80장에 대하여 추정(Estimation)한 분류 이미지의 픽셀 정확도 계산하였다. 이 실험은 training 이미지를 100장씩 추가하며 총 10회에 걸쳐 반복하고, 표 1.에 그 결과를 보였다.

실험 1의 결과는 1000장의 training 이미지를 매회 100장씩 10회 추가 학습을 수행하였을 때 계속 픽셀 정확도 증가하고, 누적 7.96% 증가함을 보인다. 이것은 ALNN의 추가 학습에서 'Forgetting' 문제가 발생하지 않음을 보이고, 이에 추가 학습의 진행에 따라 분류 성능이 향상됨을 알 수 있다.

표 1. ALNN의 추가 학습 결과.
Table 1. Result of ALNN Additional Learning

Additional learning count	Training image count	Matching rate (%)
1	100	85.53
2	200	86.41
3	300	87.83
4	400	87.86
5	500	88.88
6	600	89.58
7	700	89.76
8	800	91.51
9	900	92.49
10	1000	93.49

표 2. ANN의 추가 학습 결과
Table 2. Result of ANN Additional Learning

Additional learning count	Training image count	Matching rate (%)
1	100	84.86
2	200	74.66
3	300	74.86
4	400	86.68
5	500	80.54
6	600	88.04
7	700	86.94
8	800	90.59
9	900	88.34
10	1000	75.81

4.2 ANN 알고리즘 추가 학습

실험 2는 ALNN의 대조군으로 ANN 알고리즘을 실험 1과 동일한 절차로 수행하고, 실험 1의 결과와 비교하였다. 실험 2의 결과, 표 2.에서와 같이 추가 학습 2회의 픽셀 정확도가 추가 학습 1회의 픽셀 정확도보다 10.2% 하락하였다. 이 후 10회 추가 학습을 수행할 때 픽셀 정확도가 상승과 하락을 반복한다. 이것으로 ANN 알고리즘에 의한 추가 학습에서 기존 학습 데이터가 손실되는 'Forgetting'문제가 발생함을 알 수 있다.

그림 7.은 실험 1과 2의 분류 결과이다. 그림 7.(a)는 실험에 사용한 원이미지, (b)는 원이미지의 GroundTruth이다. 그림 7.(c)는 추가 학습이 1회 수행되었을 때 ALNN과 ANN 알고리즘에서 분류 결과 이미지이다. 그림 7.(d)는 추가 학습이 5회 수행되었을 때 두 알고리즘의 분류 결과이고, (e)는 추가 학습이 10회 수행되었을 때의 각 알고리즘의 분류 결과 이미지이다. 그림 7.에서, 추가 학습이 진행됨에 따라 ALNN에 의한 분류 이미지의 픽셀 정확도는 올라가고, ANN에 의한 분류 이미지의 픽셀 정확도는 내려가는 것을 육안으로도 확인할 수 있다.

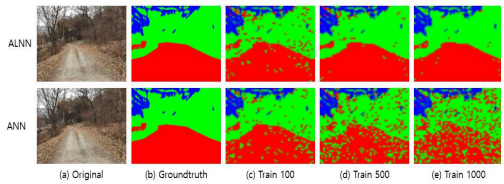

그림 7. ALNN과 ANN의 분류 결과.
Fig. 7. Result of ALNN and ANN Classification.

그림 8.은 실험1과 2의 결과를 비교한 그래프이다. 실험1의 결과는 정합률의 변동이 작고 픽셀 정확도가 올라간다. 반면에 실험 2의 결과는 픽셀 정확도의 변동이 최대 15.9%까지 발생된다. 이를 통해 ALNN 알고리즘은 추가 학습이 가능하며 'Forgetting' 문제가 발생하지 않음을 알 수 있다.

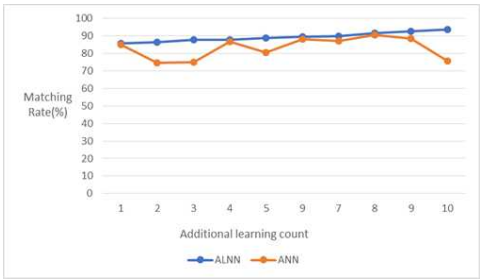

그림 8. ALNN과 ANN의 정합률.
Fig. 8. Matching rate of ALNN and ANN

4.3 ALNN 알고리즘 연산 속도 측정

실험 3에서는 ALNN 알고리즘을 GLSL 기반

GPU 병렬 처리를 하였을 때와 병렬처리를 하지 않았을 때의 연산 속도를 측정하고 비교하였다. 또 ALNN 알고리즘의 대조군으로서 ANN 알고리즘에 대하여 연산 속도를 측정하여 결과를 비교하였다. 연산 속도는 분류 속도이며, 1000장의 training 이미지로 학습하고 80장의 test 이미지에 대하여 평균 연산 속도를 구했다. 표 3.은 실험 3의 결과이다.

표 3. 연산 속도 비교.
Table 3. Comparison of computing speed.

Type	Computing Speed
ANN CPU	399.3884
ALNN CPU	330.9658
ALNN GPU	1.41

실험 3의 결과, GPU 병렬 처리를 사용하지 않는 ALNN 알고리즘의 속도가 ANN보다 69초 빨랐다. 이 차이는 ALNN과 ANN 알고리즘에서 사용된 D의 수치와 GroundTruth의 Class의 숫자에 의해 변경이 될 수 있기 때문에 큰 의미가 없다. 그러나 GPU 병렬 처리를 사용한 경우 ALNN 알고리즘의 연산 속도가 사용하지 않았을 때 보다 234배 향상된 것은 큰 의미를 갖는다. 이로써 제안한 ALNN이 빈번히 추가 학습이 필요한 국방 분야 등의 무인 차량 운행을 위한 실시간 주행로 검출에 활용될 수 있다.

5. 결론

본 논문에서는 국방 분야의 무인 차량의 비포장도로, 야지 등의 운용 환경을 고려하여 추가 학습이 가능하도록 ANN 알고리즘을 수정한 ALNN 알고리즘을 제안하였다. ALNN 알고리즘은 ANN의 최근접 데이터 선택 기법을 채용하고, 거리와 Class 비교를 통하여 학습 데이터를 추가하므로 기존의 Forgetting 문제를 해결하였다.

이를 검증하기 위해 ALNN과 ANN 알고리즘에 총 10회의 추가 학습 실험을 수행하고 그 결과를 비교함으로써 ALNN 알고리즘은 'Forgetting' 문제에 영향을 받지 않음을 확인하였다. 또 ALNN 알고리즘은 추가 학습을 수행할수록 데이터 분류 성능이 7.96% 향상됨을 보였다.

그러나, ALNN 알고리즘은 학습이 진행됨에 따라 학습 데이터가 빠르게 쌓이게 되고 이에 따라 연산 속도 저하 문제가 발생한다. 본 연구에서는 속도 저하 문제를 GLSL 기반 GPU 병렬 처리를 적용하여 기존 CPU 연산 속도보다 연산 속도를 234배 향상시키므로 해결하였다. 이에 제안한 ALNN 알고리즘은 운용 환경이 자주 바뀌어 추가 학습이 빈번히 요구되는 무인 차량의 주행로 검출에 효과적으로 사용할 수 있을 것으로 기대된다.

REFERENCES

[1] Jihong Min, Juil Sock, Jun Kim, Seongyong Ahn, Sungdae Sim and Kiho Kwak, "Perception Technology for Autonomous Navigation in Rough Terrain," *Proceedings of 2016 KIEE Summer Conference*, pp. 1462~1463, 2016.

[2] Erke Shang, Xiangjing An, Jian Li, Lei Ye1 and Hangen He, "Robust unstructured road detection: the importance of contextual information," *International Journal of Advanced Robotic Systems*, Vol. 10: 179, pp. 1-8, 2013. DOI: 10.5772/55560.

[3] Abhijit Kundu, Vibhav Vineet and Vladlen Koltun, "Feature space optimization for semantic video segmentation," *Proceedings of 2016 IEEE Conference on Co*mputer Vision and Pattern Recognition, pp. 3168-3175, 2016. DOI: 10.1109/CVPR.2016.345.

[4] Fernando D. Vázquez1, J. Salvador Sánchez and Filiberto Pla, "Learning and forgetting with local Information of new objects," *Proceedings of CIARP' 2008,* pp.261-268, 2008. DOI: 10.1007/978-3- 540-85920-8_32.

[5] Martha Roseberry and Alberto Cano, "Multi-label kNN Classifier with Self Adjusting Memory for Drifting Data Streams," *Proceedings of Machine Learning Research 2018*, pp. 1-15, 2018.

[6] Navneet Dalal and Bill Triggs, "Histograms of

oriented gradients for human detection," *Proceedings of 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 886-893, 2005. DOI: 10.1109/CVPR. 2005.177.

[7] Sunil Arya, David Mount, Nathan Netanyahu and Ruth Silverman, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *Journal of the ACM,* Vol. 45, No. 6, pp. 891-923, 1998.

[8] Dave Shreiner, OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1, Pearson Education, 2009.

[9] Dan Ginsburg and Budirijanto Purnomo, OpenGL ES 3.0 programming guide, Addison-Wesley Professional, 2014.

저자약력

구 본 우(Bon Woo Ku) **[정회원]**

•2016년 세종대학교 디지털 콘텐츠 학과 공학사
•2018년 중앙대학교 첨단영상대학원 영상공학석사
•2018년~현재 ㈜ 심네트 사원

〈관심분야〉 게임 인공지능, 패턴인식, 컴퓨터 그래픽스

김 준 겸(Jun Kyum Kim) **[정회원]**

•2006년 공주대학교 컴퓨터멀티미디어학과 공학사
•2019년 한밭대학교 컴퓨터공학 대학원 공학석사 (예정)
•2009년~현재 ㈜ 심네트 과장

〈관심분야〉 인공지능, 영상처리, 패턴인식, 컴퓨터 비전

이 은 주(Eun Joo Rhee) **[정회원]**

•1989년 충남대학교 전자공학과 공학박사
•2004년 동경공업대학 박사후연구원
•2008년 오레곤 과학기술대학원대학교 객원교수
•1989년~현재 한밭대학교 컴퓨터공학과 교수

〈관심분야〉 인공지능, 영상처리, 패턴인식, 컴퓨터 비전, 딥러닝