Journal of the Korea Industrial Information Systems Research Vol. 23 No. 4, Aug. 2018 : 41-53
http://dx.doi.org./10.9723/jksiis.2018.23.4.041 ISSN:1229-3741

전처리와 특징 추출이 CNN기반 화재 탐지 성능에 미치는 효과

(Effects of Preprocessing and Feature Extraction on CNN-based Fire Detection Performance)

이 정 환[1], 김 병 만[2], 신 윤 식[3]*
(JeongHwan Lee, Byeong Man Kim, and Yoon Sik Shin)

요 약 최근 들어 머신 러닝 기술의 발달로 기존 영상 기반의 응용시스템에 딥러닝 기술을 적용하는 사례들이 늘고 있다. 이러한 맥락에서 화재 감지 분야에서도 CNN (Convolutional Neural Network)을 적용하는 시도들이 이루어지고 있다. 본 논문에서는 기존 전처리 방법과 특징 추출 방법이 CNN과 결합되었을 때 화재 탐지에 어떤 효과를 유발하는지를 검증하기 위해 인식 성능과 학습 시간을 평가해 보았다. VGG19 CNN 구조를 변경, 즉 컨볼루션층을 조금씩 늘리면서 실험을 진행한 결과, 일반적으로 전처리하지 않는 이미지를 사용한 경우가 성능이 훨씬 좋음을 확인할 수 있었다. 또한 성능적인 측면에서는 전처리 방법과 특징 추출 방법이 부정적인 영향을 미치지만 학습 속도 측면에서는 많은 이득이 있음을 확인할 수 있었다.

핵심주제어 : 화재 탐지, 딥러닝, CNN, 전처리, 특징 추출

Abstract Recently, the development of machine learning technology has led to the application of deep learning technology to existing image based application systems. In this context, some researches have been made to apply CNN (Convolutional Neural Network) to the field of fire detection. To verify the effects of existing preprocessing and feature extraction methods on fire detection when combined with CNN, in this paper, the recognition performance and learning time are evaluated by changing the VGG19 CNN structure while gradually increasing the convolution layer. In general, the accuracy is better when the image is not preprocessed. Also it's shown that the preprocessing method and the feature extraction method have many benefits in terms of learning speed.

Key Words : Fire Detection, Deep Learning, CNN, Preprocessing, Feature Extraction.

* Corresponding Author : ysshin@kumoh.ac.kr
+ 이 논문은 2017년 금오공과대학교 연구비 지원에 의해 연구되었음.
Manuscript received June 25, 2018 / revised August 10, 2018 / accepted August 20, 2018
1) 금오공과대학교 컴퓨터소프트웨어공학과, 제1저자
2) 금오공과대학교 컴퓨터소프트웨어공학과, 제2저자
3) 금오공과대학교 컴퓨터소프트웨어공학과, 교신저자

1. 서 론

행정안전부 화재통계연보에 따르면 2016년 발생한 화재는 43,413건, 2017년에는 44,178건으로 수년간 발생 빈도를 보이고 있다 [1]. 최근 들어, 서문시장 화재, 소래포구 종합어시장 화재, 고령 양산 공장 화재, 제천 스포츠 센터 화재 등 많은 피해를 주는 화재 사건이 발생했다. 특히 겨울철에 많은 화재 사건이 발생했다. 이러한 화재는 주로 인간의 실수 또는 시스템의 고장으로 인해 발생하며, 이로 인해 인간의 생명을 위험에 빠뜨릴 뿐만 아니라 경제적인 피해도 유발한다. 이러한 피해 사례를 고려할 때 화재를 조기에 감지하는 것이 무엇보다도 중요하다. 이러한 맥락에서 화재를 조기에 자동으로 감지하기 위한 많은 연구들이 진행되어 왔다.

화재 자동 감지 방법은 화재 센서(연기, 불꽃, 온도 등) 기반 방법과 카메라를 이용한 영상 처리 기반 방법으로 구분할 수 있다 [2]. 센서 기반의 화재 감지는 주변 환경의 여러 요인에 따라 시스템의 성능이 크게 영향을 받게 되는데, 자외선 검출 기술을 사용하여 불꽃을 검출할 경우, 연기나 기타 요인에 인해 자외선이 흡수될 수 있고 이로 인해 민감도가 떨어질 수 있다. 온도 센서는 계속 사용하게 되면 시간이 지날수록 변하게 되고 (특히 고온에서 사용되는 경우) 측정 오류를 초래하게 된다. 특히 넓은 지역에 대한 감지가 필요할 경우 일정한 간격으로 여러 대를 설치해야 하는 비용적인 문제가 발생한다 [3]. 연기 센서는 온도 센서에 비해 화재 발생 초기에 감지할 수 있는 장점은 있지만 설치 장소에 따라 성능이 많이 달라지며 먼지나 담배 연기와 같은 요인에도 반응할 수 있다.

영상 기반의 화재 감지는 기존의 감시 카메라의 재활용으로 인한 비용 절감, 넓은 지역을 감시할 수 있는 점, 열확산 대기 시간이 없어 비교적 빠르게 반응한다는 점, 방문하지 않고도 화재를 확인할 수 있어 오인 출동을 줄일 수 있다는 점, 특정 매개 변수의 조정을 통해 연기 및 화염을 감지 할 수 있는 유연성, 그리고 화재 크기, 위치 및 연소 정도와 같은 화재 세부 정보를 얻을 수 있는 점 등 여러 장점으로 인해 많은 연구자들의 관심을 끌었으며 결과적으로 시각적 특징을 이용한 여러 화재 탐지 방법 [4,5,6,7,8,9]이 제안되었으며 좋은 성능을 보였다. 그러나 영상 기반 방법도 화재와 유사한 사람과 물체로 인한 오인식, 그리고 그림자, 빛 반사 및 깜박임과 같은 불규칙한 조명과 캡쳐 이미지의 좋지 않는 화질 등으로 인한 성능 저하의 문제점들을 갖고 있다 [10].

최근 들어 머신 러닝 기술의 발달로 기존 영상 기반의 응용시스템에 딥러닝 기술을 적용하는 사례들이 많아지고 있다. 화재 감지 분야에서도 영상 처리에 좋은 성능을 보이는 딥러닝의 지도학습(Supervised Learning) 모델 가운데 하나인 CNN (Convolutional Neural Network)을 적용하는 시도들 [2,10,11,12]이 이루어지고 있다. 전통적인 인공신경망을 이용한 영상 처리에서는 입력의 크기를 줄이기 위해 특징 추출 알고리즘에 따라 특징을 추출하거나 전처리 과정을 통해 후보 영역을 선별하는 방법을 사용한다. 반면에 CNN은 보통 이러한 특징 추출이나 전처리 과정 없이 원시 이미지를 그대로 입력 받아 학습을 통하여 학습 데이터에 맞는 특징들을 추출하여 사용하게 된다. 과연 기존의 전처리 방법이나 특징 추출을 CNN과 접목시키면 어떤 효과가 있을까? 본 논문은 바로 이러한 의문에서부터 출발하였다. 이러한 의문을 해소하기 위해 본 문문에서는 화재 인식 시 사용되었던 전처리 방법 하나 [13]와 이미지 특징 중 하나인 HOG 특징 [14]을 CNN과 결합 할 때 인식 성능 측면과 학습 시간 측면에서 비교하여 보았다.

본 논문의 구성은 다음과 같다. 2장에서는 CNN 기본 사항과 CNN을 이용한 화재 탐지 방법 그리고 본 논문에서 사용할 전처리 방법과 HOG 특징에 대해서 살펴본다. 3장에서는 본 논문에서 제안하는 방법을, 4장에서는 데이터 수집 및 실험환경 그리고 실험 결과에 대해 살펴본다. 5장에서 결론과 향후 연구계획으로 끝을 맺는다.

2. 관련연구

2.1 CNN (Convolutional Neural Network)

기존 인공신경망은 일반적으로 입력층, 은닉층, 그리고 출력층으로 구성되며, 입력층과 은닉층, 은닉층과 은닉층, 은닉층과 출력층 사이는 완전 연결 (Fully Connected)된 구조를 이루고 있다. 은닉층이 있는 신경망을 다층퍼셉트론 (MLP : Multi Layer Perceptron)이라 부르며 역전파 (Backpropagation) 학습알고리즘을 이용하여 학습한다. MLP가 제안될 당시에도 은닉층을 깊게 할 수 있었으나 Gradient Vanishing 문제나 과적합 (Overfitting) 문제들로 인해 깊은 층에 대한 학습이 제대로 이루어지지 않아 하나 또는 두 개 정도의 은닉층을 주로 사용하였다. 그 이후 꾸준히 깊은 층 학습 시 발생 가능한 문제들을 해결할 수 있는 기술들이 제안이 되고 최근 들어 많은 응용에서 두 층 이상의 은닉층을 사용하는 모델들이 좋은 성능을 보이면서 딥러닝이 기계학습의 핵심 기술로 부상하게 되었다. 깊은 은닉층과 이에 맞는 학습 방법을 사용하는 신경망 구조를 DNN (Deep Neural Network)라 한다.

영상처리 분야에서도 꾸준히 신경망 모델을 이용하는 연구들이 진행되어 왔는데, 이 중 1980년도에 제안된 네오컨그니트론 (Neocongnitron)은 인간의 시각 피질을 모방하여 그 가능성을 보였다. 네오컨그니트론은 계층적인 구조로 이루어져 있는데, 하위 층의 뉴런들은 인식 대상의 저수준 특징들을 인지하고, 그 다음 층은 이전 층의 특징들을 이용하여 좀 더 높은 수준의 특징들을 인지하는 구조로 이루어져 있다. 이러한 구조가 지속적으로 발전되어 오늘날의 CNN에 이르게 되었다.

CNN은 입력층, 은닉층 그리고 출력층에 컨볼루션층 (Convolutional Layer)과 풀링층 (Pooling Layer)을 추가한 구조를 띠고 있다. 여러 개의 컨볼루션층이 있고 컨볼루션층은 계층적이어서 상위 컨볼루션층은 하위 컨볼루션층의 출력을 이용하여 좀 더 높은 수준의 특징들을 추출하게 된다. 각 컨볼루션층은 필터를 이용하여 특징을 추출하게 되는데, 이 필터는 고정된 것이 아니라 학습에 의해 학습 데이터에 맞게 결정이 된다. 필터는 초기에 랜덤값으로 초기화 되고 기존 다층신경망 학습 과정을 통해 데이터에 맞게 재조정되게 된다. 필터 하나가 하나의 특징에 대응되며 이 필터를 이미지에 적용하면 특징값들, 즉 특징맵 (Feature Map)을 얻게 된다. 이 특징맵이 컨볼루션층의 출력이 되는데, 보통 한 컨볼루션층은 여러 필터들로 구성되어 있어 결국 특징맵 집합이 한 컨볼루션층의 출력이 되게 된다. 풀링층은 컨볼루션층 사이에 위치해 특징맵의 사이즈를 줄이는 역할과 위치 변화에 덜 민감하도록 하는 역할을 담당한다.

CNN은 보통 앞단에 컨볼루션층과 풀링층이 있고 뒷단에 기존의 신경망과 같은 완전 연결층이 있는 형태로 이루어져 있다. 기존 영상 처리에서 특징 추출 알고리즘에서 담당하던 일을 앞단에서 자동으로 수행하며, 따라서 주로 앞단의 구조에 따라 성능이 달라진다. 현재까지 ReNet-5, AlexNet, GooLeNet, REsNet, VGG19 [15] 등 다양한 구조의 CNN들이 제안되어 왔고 계속 제안되고 있는데 점점 더 깊어지는 추세이다. 본 논문에서는 CNN 구조가 실험 목적에는 큰 영향을 주지 않기 때문에 CNN 구조 중에서 간단하고 직관적이면서도 좋은 성능을 보여준 VGG19를 선택하여 이를 변경하며 실험을 진행하였다.

2.2 CNN을 이용한 화재 탐지

논문 [11]에서는 기존 화재 탐지 방법인 규칙 기반이나 특징 벡터 기반의 탐지 방법이 그 규칙이나 특징을 추출하기가 쉽지 않으며 설령 추출하더라도 전문가가 경험한 화재의 특징에 의존할 수밖에 없으며 따라서 성능 저하를 초래한다고 지적하고, 이에 대한 대안으로 특징을 자동으로 추출할 수 있는 CNN 기반의 방법을 제안하고 실 비디오 시퀀스에 대해 그 방법이 유용함을 보였다.

논문 [2]에서는 입력 영상 프레임으로부터 색상정보를 이용하여 화염의 후보 영역을 먼저 검출한 다음, 학습된 CNN(Convolutional Neural Network)을 활용해서 최종적으로 화재를 감지하는, CNN을 활용한 영상 기반의 화재 감지 방법을 제안하였다. 후보 영역을 대상으로 하기 때문에 이미지의 크기가 작아져서 학습속도가 빠르며 하나의 객체만을 대상하기 때문에 성능이 높아지는 장점이 있다. 반면에 후보 영역을 검출하고

그 각각에 대한 레이블을 작성해 주어야 하는 번거로움이 있다.

논문 [10]에서는 CCTV 감시 카메라에 적합하게 만들어진 CNN을 활용한 조기 화재 탐지 프레임 워크를 제안했다. 여기서는 다양한 실내 및 실외 환경에서 화재를 감지 할 수 있도록 CNN을 구성하였으며, 실험을 통해 기존 방법에 비해 화재 탐지 정확성이 높아지고 효과적으로 감지할 수 있음을 보였다.

논문 [12]에서는 기존의 CNN 기반 화재 탐지 방법들이 실제 환경 하에서는 성능이 저하됨과 그 이유가 학습 및 검증 데이터가 실 데이터가 아니라 인위적으로 잘 구성된 데이터 셋을 사용하기 때문이라 지적하면서 이를 해결하기 위해 기존 VGG16과 ResNet50의 기 학습된 모델을 활용하여 완전 연결층을 통해 미세 조정하는 더 깊은 CNN을 제안하였다. 그리고 실제 환경에서의 성능을 측정하기 위해 불균형 데이터 셋, 즉, 분류가 어려운 비화재 이미지를 훨씬 많이 포함한 이미지 셋을 테스트용 데이터로 사용하였다. 전반적으로 제안 CNN이 더 어려운 데이터 세트에서 우수한 성능을 보임을 실험을 통해 보였다.

2.3 전처리 방법

논문 [13]에서는 RGB칼라 모델이 아닌 YCbCr 칼라 모델에서 화재 영역 검출 알고리즘을 제안하였다. 해당 논문은 화재의 경계 추출, 노이즈 제거를 통해 화재 영역을 뚜렷하게 검출하였다. 본 논문에서는 [13]에서 이용한 YCbCr칼라 공간에서 각 성분간의 칼라 정보를 이용하여 화재 영역을 검출하는 [식 1]을 적용하여 합성 사진과 일반 거리사진을 전처리 하였다.

$$F(x,y) = G(x,y) \times H(x,y)$$

$$G(x,y) = \begin{cases} 1 \text{ if } |C_r - C_b| \geq \gamma \\ 0 \; otherwise \end{cases} \qquad [식 1]$$

$$H(x,y) = \begin{cases} 1 \text{ if } Y \geq \alpha \\ 0 \; otherwise \end{cases}$$

Fig. 1의 왼쪽은 합성 사진이며, 오른쪽 이미지는 [식 1]을 적용한 후의 이미지이다.

Fig. 1 A Fire Image and its Preprocessed Image

2.4 호그 (HOG) 특징

HOG(Histogram of Oriented Gradients)는 Dalal 등에 의해 제안된 특징 추출 방법으로, 슬라이딩 윈도우를 기반으로 기울기의 분포를 계산하여 특징을 추출한다. 영상에서의 기울기란 영상 내에서 밝기가 변화하는 방향을 의미한다. HOG는 영상 내의 지역별 영역의 기울기 방향 성분의 발생 빈도를 누적시켜 히스토그램을 생성하며, 배경과 객체간의 대비(Contrast)와 잡음(Noise)에 강인함이 검증되어 있으므로 정확한 객체의 특징 추출을 필요로 하는 분야에 주로 이용된다[14]. 영상을 특정 Cell크기로 나누어서 각 Cell마다 픽셀의 기울기를 구한다. 기울기에 해당하는 값을 특정 개수의 Bin으로 나누어 각 Bin에 대한 히스토그램을 구한다. 본 논문에서는 Cell의 크기를 4x4로 설정하여 256x256 흑백 이미지를 나누었으며 각 Cell당 9개의 Bin을 두어 히스토그램을 만들었다. 이때 파이썬 Scikit-Image 라이브러리의 Feature 모듈의 Hog 함수를 사용하였다.

3. 제안 방법

본 논문에서는 일반 거리사진과 불 사진을 합성한 RGB이미지와 [13]에서 제안한 화재 검출식 [식 1]을 이용한 전처리된 흑백 이미지, HOG 알고리즘을 적용한 호그 특징을 각각 훈련 데이터로 사용했다. RGB이미지는 Red, Green, Blue를 조합하여 각 픽셀값을 표현한 이미지로 픽셀마다 총 3개의 채널을 가진다. 칼라 이미지는 가로, 세로의 크기가 256x256되도록 리사이징 하였다. 전

처리 (Preprocessed) 이미지는 Gray Scale만 표현하면 되므로 총 1개의 채널을 가진다. 호그 특징은 흑백 이미지를 4x4 크기의 Cell로 나누어 각 cell의 픽셀 방향을 계산하여 방향의 범위에 따라 9개의 bin에 누적시켰다. 따라서 한 이미지 당 64 x 64개의 cell이 생성되며 Cell 당 9개의 특징값이 나오는데 이를 크기가 64x64이며 채널수가 9개인 이미지로 해석하였다. 이렇게 구성된 이미지를 본 논문에서는 호그 (HOG) 이미지라 한다. 각 이미지의 크기를 정리하면 Table 1과 같다.

Table 1 Image Size by Image Type

Image Type	Image Size
RGB	(256, 256, 3)
Preprocessed	(256, 256, 1)
Hog	(64, 64, 9)

딥러닝 모델을 구성하여 그 성능을 평가해 보았는데, 첫 번째 방법은 Table 2와 같은 3계층 완전연결 NN을 구성하여 학습시켜보았다. 표에서 “출력” 필드는 상위 레이어에서 입력받은 값을 필터를 통과시켜 얻은 출력값의 형태를 말한다. 이미지를 입력해주기 위해 3차원 형태 (가로 길이 256, 세로 길이 256, 채널 3개)을 가지는 이미지를 1차원 (256x256x3 = 196608)으로 만들어 주었다. 입력층의 입력은 사용했던 RGB이미지, 전처리 이미지, 호그 이미지 크기에 따라 달라진다.

Table 2 DNN Structure (Model1)

Image Type	Layer	Input	Output
RGB	Input	(196608)	(196608)
	dense	(196608)	(64)
	dense	(64)	(64)
	output	(64)	(2)
Preprocessed	Input	(65536)	(65536)
	Dense	(65536)	(64)
	Dense	(64)	(64)
	output	(64)	(2)
HOG	Input	(36864)	(36864)
	Dense	(36864)	(64)
	Dense	(64)	(64)
	output	(64)	(2)

두 번째 방법은 Table 3과 같은 CNN을 이용하여 학습시켜보았다. 입력 이미지에 따라 입력층의 크기가 바뀌며 이에 따라 뒤 따르는 모든 층의 입력 형태 및 출력 형태가 달라진다. 단, 모델의 레이어 구조는 동일하다. 뒤에 소개되는 모델에 대해서도 동일하게 적용된다. 아래 표에서, I는 입력층을, C는 콘볼루션층을, P는 풀링층을, D는 덴스층을, D*는 Dropout이 적용된 덴스층을, O는 출력층을 의미한다.

Table 3 Structure of Model2

RGB Image			
Layer	Filter	Input	Output
I		(256, 256, 3)	(256, 256, 3)
C	(3, 3, 16)	(256, 256, 3)	(256, 256, 16)
C	(3, 3, 16)	(256, 256, 16)	(256, 256, 16)
P		(256, 256, 16)	(128, 128, 16)
C	(3, 3, 32)	(128, 128, 16)	(128, 128, 32)
C	(3, 3, 32)	(128, 128, 32)	(128, 128, 32)
P		(128, 128, 32)	(64, 64, 32)
C	(3, 3, 64)	(64, 64, 32)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
P		(64, 64, 64)	(32, 32, 64)
D*		(65536)	(512)
D*		(512)	(512)
O		(512)	(2)
Preprocessed Image			
I		(256, 256, 1)	(256, 256, 1)
C	(3, 3, 16)	(256, 256, 1)	(256, 256, 16)
C	(3, 3, 16)	(256, 256, 16)	(256, 256, 16)
P		(256, 256, 16)	(128, 128, 16)
C	(3, 3, 32)	(128, 128, 16)	(128, 128, 32)
C	(3, 3, 32)	(128, 128, 32)	(128, 128, 32)
P		(128, 128, 32)	(64, 64, 32)
C	(3, 3, 64)	(64, 64, 32)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
P		(64, 64, 64)	(32, 32, 64)
D*		(65536)	(512)
D*		(512)	(512)
O		(512)	(2)
Hog Image			
I		(64, 64, 9)	(64, 64, 9)
C	(3, 3, 16)	(64, 64, 9)	(64, 64, 16)
C	(3, 3, 16)	(64, 64, 16)	(64, 64, 16)
P		(64, 64, 16)	(32, 32, 16)
C	(3, 3, 32)	(32, 32, 16)	(32, 32, 32)

C	(3, 3, 32)	(32, 32, 32)	(32, 32, 32)
P		(32, 32, 32)	(16, 16, 32)
C	(3, 3, 64)	(16, 16, 32)	(16, 16, 64)
C	(3, 3, 64)	(16, 16, 64)	(16, 16, 64)
C	(3, 3, 64)	(16, 16, 64)	(16, 16, 64)
C	(3, 3, 64)	(16, 16, 64)	(16, 16, 64)
P		(16, 16, 64)	(8, 8, 64)
D*		(4096)	(1024)
D*		(1024)	(1024)
O		(1024)	(2)

입력층을 통해 가로 크기 256, 세로 크기 256, 채널 개수 3(RGB)인 이미지를 입력받는다. “필터” 필드는 콘볼루션 커널의 형태로 (가로, 세로, 커널의 개수)를 말한다. 콘볼루션층에서 해당 필터 칸에 해당하는 커널을 이용하여 특징을 추출한다. 풀링층은 콘볼루션층에서 뽑아낸 특징을 간추리는 역할을 담당한다. 덴스층을 이용해 뽑아낸 특징을 모두 연결하여 뉴럴 네트워크 모델을 만들었다. 활성화 함수는 깊은 신경망에 적절한 Relu (Fig. 2)를 사용하여 학습 도중 비용함수의 미분한 값이 사라지는 Vanishing Gradient를 방지하도록 했다.

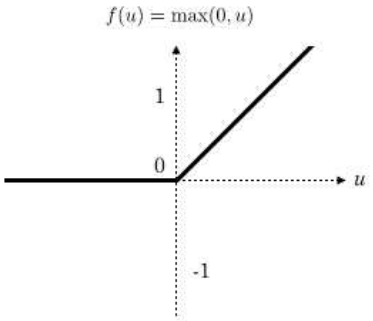

Fig. 2 Relu Function

마지막 덴스층, 즉 출력층에서 Softmax ([식 2])를 이용하여 각각의 결과 값 비율, 즉 클래스 가능성을 나타내도록 했다. 또한 풀링층을 덴스층에 이어주기 위해 특징들을 1차원 형태로 바꿔주었다.

$$Softmax(y_i) = \frac{e^{y_j}}{\sum_{k=1}^{K} e^{y_k}} \quad [식\ 2]$$

세 번째 모델은 두 번째 모델처럼 더 많은 특징을 추출하기 위해 콘볼루션층을 늘렸다. 그 구조는 Table 4와 같다.

Table 4 Structure of Model3

RGB Image			
Layer	Filter	Input	Output
I		(256, 256, 3)	(256, 256, 3)
C	(3, 3, 16)	(256, 256, 3)	(256, 256, 16)
C	(3, 3, 16)	(256, 256, 16)	(256, 256, 16)
P		(256, 256, 16)	(128, 128, 16)
C	(3, 3, 32)	(128, 128, 16)	(128, 128, 32)
C	(3, 3, 32)	(128, 128, 32)	(128, 128, 32)
P		(128, 128, 32)	(64, 64, 32)
C	(3, 3, 64)	(64, 64, 32)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
P		(64, 64, 64)	(32, 32, 64)
C	(3, 3, 128)	(32, 32, 64)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
P		(32, 32, 128)	(16, 16, 128)
D*		(32768)	(1024)
D*		(1024)	(1024)
O		(1024)	(2)
Processed Image			
I		(256, 256, 1)	(256, 256, 1)
C	(3, 3, 16)	(256, 256, 1)	(256, 256, 16)
C	(3, 3, 16)	(256, 256, 16)	(256, 256, 16)
P		(256, 256, 16)	(128, 128, 16)
C	(3, 3, 32)	(128, 128, 16)	(128, 128, 32)
C	(3, 3, 32)	(128, 128, 32)	(128, 128, 32)
P		(128, 128, 32)	(64, 64, 32)
C	(3, 3, 64)	(64, 64, 32)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
P		(64, 64, 64)	(32, 32, 64)
C	(3, 3, 128)	(32, 32, 64)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
P		(32, 32, 128)	(16, 16, 128)
D*		(32768)	(1024)
D*		(1024)	(1024)
O		(1024)	(2)
Hog Image			
I		(64, 64, 9)	(64, 64, 9)
C	(3, 3, 16)	(64, 64, 9)	(64, 64, 16)

C	(3, 3, 16)	(64, 64, 16)	(64, 64, 16)
P		(64, 64, 16)	(32, 32, 16)
C	(3, 3, 32)	(32, 32, 16)	(32, 32, 32)
C	(3, 3, 32)	(32, 32, 32)	(32, 32, 32)
P		(32, 32, 32)	(16, 16, 32)
C	(3, 3, 64)	(16, 16, 32)	(16, 16, 64)
C	(3, 3, 64)	(16, 16, 64)	(16, 16, 64)
C	(3, 3, 64)	(16, 16, 64)	(16, 16, 64)
C	(3, 3, 64)	(16, 16, 64)	(16, 16, 64)
P		(16, 16, 64)	(8, 8, 64)
C	(3, 3, 128)	(8, 8, 64)	(8, 8, 128)
C	(3, 3, 128)	(8, 8, 128)	(8, 8, 128)
C	(3, 3, 128)	(8, 8, 128)	(8, 8, 128)
C	(3, 3, 128)	(8, 8, 128)	(8, 8, 128)
P		(8, 8, 128)	(4, 4, 128)
D*		(2048)	(1024)
D*		(1024)	(1024)
O		(1024)	(2)

마지막 네번째 모델도 더 많은 특징을 추출하기 위해 콘볼루션층을 늘렸다. 그 구조는 Table 5와 같다.

Table 5 Structure of Model4

RGB Image			
Layer	Filter	Input	Output
I		(256, 256, 3)	(256, 256, 3)
C	(3, 3, 16)	(256, 256, 3)	(256, 256, 16)
C	(3, 3, 16)	(256, 256, 16)	(256, 256, 16)
P		(256, 256, 16)	(128, 128, 16)
C	(3, 3, 32)	(128, 128, 16)	(128, 128, 32)
C	(3, 3, 32)	(128, 128, 32)	(128, 128, 32)
P		(128, 128, 32)	(64, 64, 32)
C	(3, 3, 64)	(64, 64, 32)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
P		(64, 64, 64)	(32, 32, 64)
C	(3, 3, 128)	(32, 32, 64)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
P		(32, 32, 128)	(16, 16, 128)
C	(3, 3, 128)	(16, 16, 128)	(16, 16, 128)
C	(3, 3, 128)	(16, 16, 128)	(16, 16, 128)
C	(3, 3, 128)	(16, 16, 128)	(16, 16, 128)
C	(3, 3, 128)	(16, 16, 128)	(16, 16, 128)
P		(16, 16, 128)	(8, 8, 128)
D*		(8192)	(1024)
D*		(1024)	(1024)
O		(1024)	(2)
Preprocessed Image			
I		(256, 256, 1)	(256, 256, 1)
C	(3, 3, 16)	(256, 256, 1)	(256, 256, 16)
C	(3, 3, 16)	(256, 256, 16)	(256, 256, 16)
P		(256, 256, 16)	(128, 128, 16)
C	(3, 3, 32)	(128, 128, 16)	(128, 128, 32)
C	(3, 3, 32)	(128, 128, 32)	(128, 128, 32)
P		(128, 128, 32)	(64, 64, 32)
C	(3, 3, 64)	(64, 64, 32)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
C	(3, 3, 64)	(64, 64, 64)	(64, 64, 64)
P		(64, 64, 64)	(32, 32, 64)
C	(3, 3, 128)	(32, 32, 64)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
C	(3, 3, 128)	(32, 32, 128)	(32, 32, 128)
P		(32, 32, 128)	(16, 16, 128)
C	(3, 3, 128)	(16, 16, 128)	(16, 16, 128)
C	(3, 3, 128)	(16, 16, 128)	(16, 16, 128)
C	(3, 3, 128)	(16, 16, 128)	(16, 16, 128)
C	(3, 3, 128)	(16, 16, 128)	(16, 16, 128)
P		(16, 16, 128)	(8, 8, 128)
D*		(8192)	(1024)
D*		(1024)	(1024)
O		(1024)	(2)
HOG Image			
I		(64, 64, 9)	(64, 64, 9)
C	(3, 3, 16)	(64, 64, 9)	(64, 64, 16)
C	(3, 3, 16)	(64, 64, 16)	(64, 64, 16)
P		(64, 64, 16)	(32, 32, 16)
C	(3, 3, 32)	(32, 32, 16)	(32, 32, 32)
C	(3, 3, 32)	(32, 32, 32)	(32, 32, 32)
P		(32, 32, 32)	(16, 16, 32)
C	(3, 3, 64)	(16, 16, 32)	(16, 16, 64)
C	(3, 3, 64)	(16, 16, 64)	(16, 16, 64)
C	(3, 3, 64)	(16, 16, 64)	(16, 16, 64)
C	(3, 3, 64)	(16, 16, 64)	(16, 16, 64)
P		(16, 16, 64)	(8, 8, 64)
C	(3, 3, 128)	(8, 8, 64)	(8, 8, 128)
C	(3, 3, 128)	(8, 8, 128)	(8, 8, 128)
C	(3, 3, 128)	(8, 8, 128)	(8, 8, 128)
C	(3, 3, 128)	(8, 8, 128)	(8, 8, 128)
P		(8, 8, 128)	(4, 4, 128)
C	(3, 3, 128)	(4, 4, 128)	(4, 4, 128)
C	(3, 3, 128)	(4, 4, 128)	(4, 4, 128)
C	(3, 3, 128)	(4, 4, 128)	(4, 4, 128)
C	(3, 3, 128)	(4, 4, 128)	(4, 4, 128)
P		(4, 4, 128)	(2, 2, 128)
D*		(512)	(1024)
D*		(1024)	(1024)
O		(1024)	(2)

4. 실험 및 성능 평가

4.1 데이터 수집 및 실험 환경

Python의 HTTP 요청 처리를 위한 Requests 라이브러리, HTML 파싱을 쉽게 할 수 있도록 해주는 BeautifulSoup 라이브러리와 인터넷의 파일을 쉽게 다운받도록 해주는 Wget 라이브러리를 이용하여 무료 이미지 제공 사이트 Pixabay에서 웹 크롤링을 통해 화재 사진, 거리 사진을 얻었다. 얻은 화재 및 거리 사진을 합성하여 Fig. 3과 같은 합성 사진을 만들었다.

Fig. 3 Composite Image of Street and Fire

합성 방식은 크롤링으로 얻은 거리 사진 762개를 좌우 반전시켜 총 1524개의 거리사진으로 만든 후 검은색 배경 불 사진 201개를 반복시키며 합성시켰다. 이렇게 해서 거리 사진 1524개와 합성 사진 1524개를 만들었다. 합성 시 불 사진의 크기를 거리 사진의 20~50%로 랜덤 스케일링 후 불 사진 전체가 들어갈 수 있는 위치에 랜덤으로 위치시켰다.

트레이닝 및 테스트에 사용한 컴퓨터 사양은 Table 6과 같다. 사용한 개발 언어는 Python 3.5 이며, 구글에서 제공해주는 딥러닝 오픈소스 라이브러리 Tensorflow와 Tensorflow Wrapper인 Keras를 사용하여 머신러닝을 수행하였다. 학습 시 사용한 학습율은 1e-5이며, 배치의 크기는 32 그리고 Optimizer는 Adam Optimizer를 사용하였다. 성능 평가 시 수집한 데이터를 학습용과 테스트용으로 랜덤으로 분리하여 사용하였다. 즉, 화재 1524 개의 이미지 중 1220개를 학습용으로, 304개를 테스트용으로, 그리고 거리사진 1524 중 1220개를 학습용으로, 304개를 테스트용으로 사용하였다.

Table 6 Computer Specifications

운영체제	Windows 7 Ultimate K
메모리(RAM)	4.00GB
GPU	NVIDIA GeForce GTX 970

4.2 실험 결과

학습 셋을 이용하여 반복횟수만큼 학습한 후 학습 셋에 대한 정확도와 테스트 셋에 대한 정확도를 살펴보았다. 학습용 데이터와 테스트용 데이터을 어떻게 분리하는가에 따라 성능이 달라질 수 있어 랜덤으로 분리하는 작업을 10번 수행하였고 그 각각의 결과를 평균하는 방법으로 성능을 측정하였다. 전처리하지 않은 이미지를 이용하여 첫번째 모델을 훈련하면서 기록한 학습 셋과 테스트 셋의 정확도는 Fig. 4와 같다. 전처리된 이미지를 이용하여 모델1을 훈련하면서 기록한 학습 셋과 테스트 셋의 정확도는 Fig. 5와 같다. 호그 이미지를 이용하여 모델1을 훈련하면서 기록한 학습 셋과 테스트 셋의 정확도는 Fig. 6과 같다. x 축은 epoch이며 y축은 정확도이다.

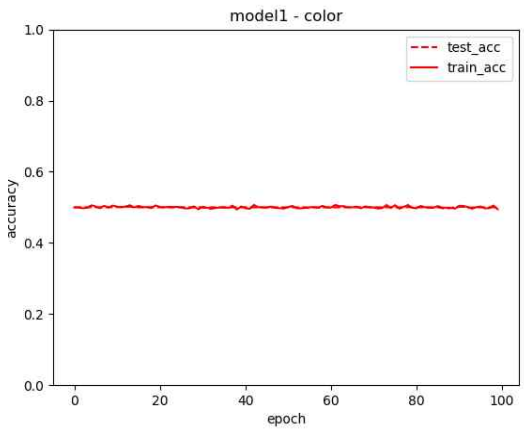

Fig. 4 Performance of Model1 for RGB Image

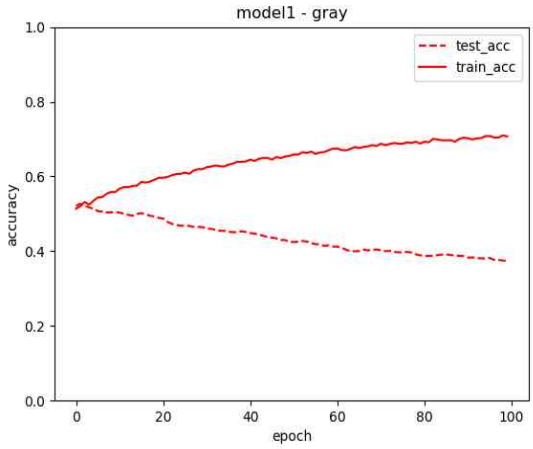

Fig. 5 Performance of Model1 for Preprocessed Image

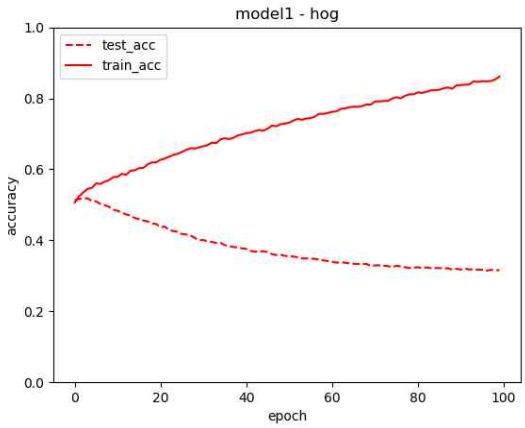

Fig. 6 Performance of Model1 for HOG Image

그림에서 보듯이 원 이미지에 대한 학습 셋 성능과 테스트 셋 성능은 0.5로 학습이 제대로 되지 않음을 알 수 있다. 거의 무작위 수준이라고 볼 수 있다. 원 이미지에 대해 이런 결과는 입력 벡터의 차원이 너무 커서 제대로 학습이 되지 않는 것으로 보인다. 전처리 이미지는 횟수를 반복할수록 학습 셋의 성능은 증가하고 반대로 테스트 셋의 성능은 감소하는 것으로 보아 반복할수록 과적합 (Overfitting)이 발생하는 것으로 보인다. 호그 이미지도 전처리 이미지와 비슷한 결과를 보이고 있다.

두 번째 모델에 대한 정확도는 Fig. 7, Fig. 8, Fig. 9와 같다. 원 이미지에 대해서는 반복횟수가 증가할수록 학습 셋의 성능은 증가하지만 테스트 셋의 성능은 0.5 수준으로 거의 일정한 것으로 보아 과적합이 초반에 발생하고 이후 학습된 내용이 테스트 셋에 거의 영향을 미치지 못함을 알 수 있다. 하지만 전처리 이미지에 대한 성능은 모델1과 비슷하게 반복할수록 과적합이 심화되어 학습 셋에 대한 성능과 테스트 셋에 대한 성능이 벌어짐을 알 수 있다. 호그 이미지인 경우, 반복횟수가 증가할수록 학습 셋 성능이 조그씩 향상이 되고 있고 테스트 셋과의 차이도 그리 크게 나지 않고 있다.

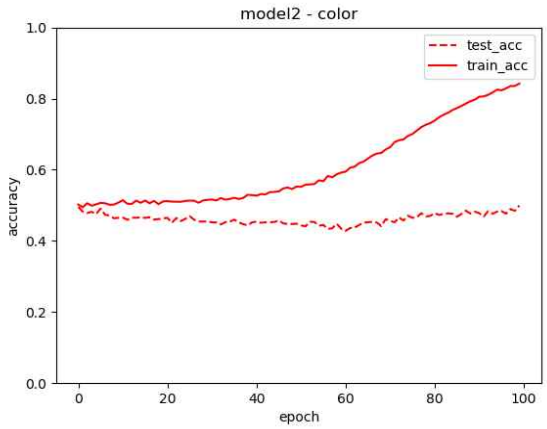

Fig. 7 Performance of Model2 for RGB Image

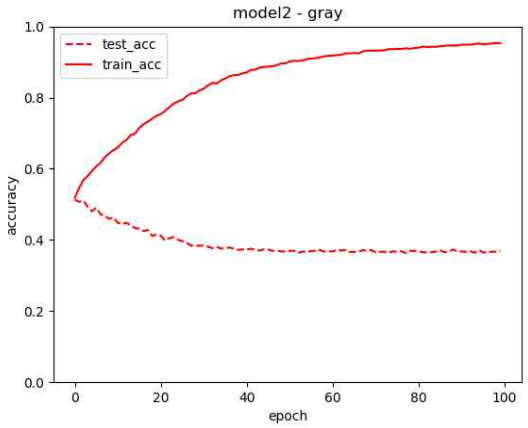

Fig. 8 Performance of Model2 for Preprocessed Image

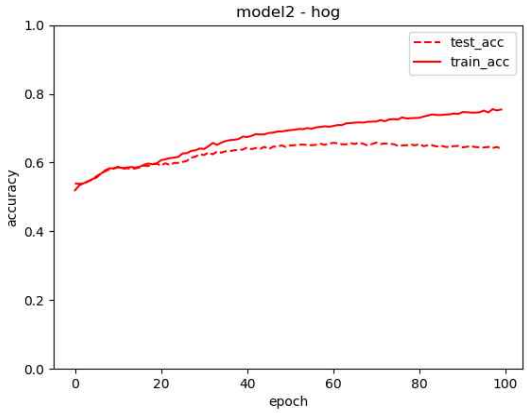

Fig. 9 Performance of Model2 for HOG Image

세 번째 모델에 대한 정확도는 Fig. 10, Fig. 11, Fig. 12와 같다. 원 이미지에 대한 성능은 모델2에 비해 상당히 좋아지는 것으로 보인다. 하지만 반복 횟수 30을 기점으로 테스트 셋에 대한 성능이 포화되는 것으로 보인다. 반복 횟수 30 이후에도 학습 셋에 대한 성능이 증가하는 것으로 보아 반복 횟수 30을 기점으로 오버피팅이 발생하는 것으로 보인다. 전처리 이미지에 대한 성능은 큰 흐름이 모델2와 비슷하다. 단, 모델2인 경우는 초반부터 학습 셋과 테스트 셋의 성능 차이가 크나 모델3인 경우는 반복 횟수 10 이후에 차이가 커짐을 알 수 있다. 호그 이미지인 경우, 모델2와 비슷하게 반복 횟수가 증가할수록 학습 셋 성능이 조그씩 향상이 되고 있고 테스트 셋과의 차이도 그리 크게 나지 않고 있다. 전반적으로 모델2에 비해 성능 향상이 되었다.

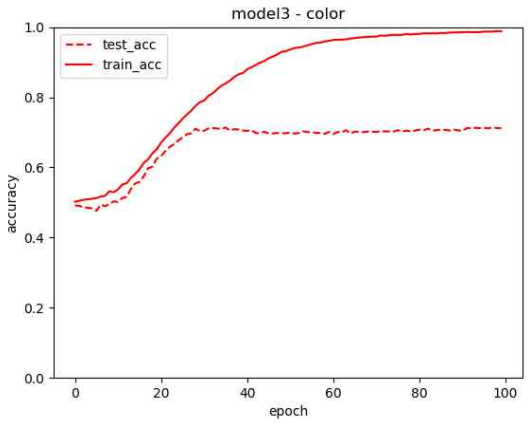

Fig. 10 Performance of Model3 for RGB Image

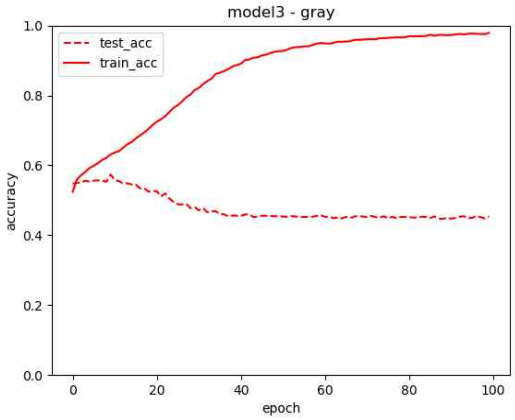

Fig. 11 Performance of Model3 for Preprocessed Image

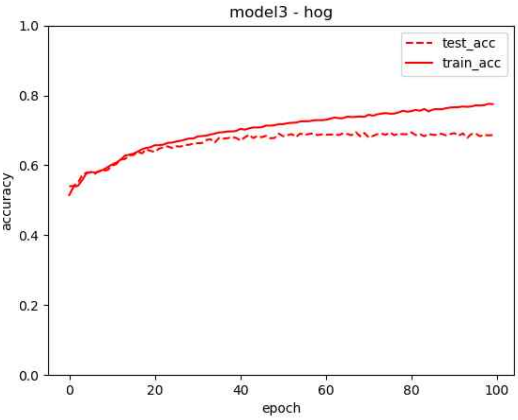

Fig. 12 Performance of Model3 for HOG Image

마지막 모델에 대한 정확도는 Fig. 13, Fig. 14, Fig. 15와 같다. 원 이미지에 대한 성능은 모델3에 비해 상당히 좋아지는 것으로 보인다. 하지만 반복 횟수 30을 기점으로 테스트 셋에 대한 성능이 포화되는 것으로 보인다. 전처리 이미지에 대한 성능은 모델3와 비슷해 보인다. 단, 반복 횟수 30을 기점으로 그 차이가 커지는 것으로 보인다. 호그 이미지는 모델2 및 모델3과 비슷하게 반복 횟수가 증가할수록 학습 셋 성능이 조그씩 향상이 되고 있고 테스트 셋과의 차이도 그리 크게 나지 않고 있다. 단 반복 횟수가 증가할수록 테스트 셋 성능이 조금씩 감소하는 것으로 보아 과적합이 조금씩 진행되는 것으로 보인다.

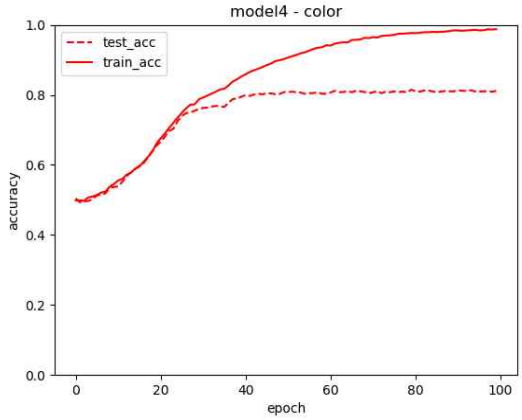

Fig. 13 Performance of Model4 for RGB Image

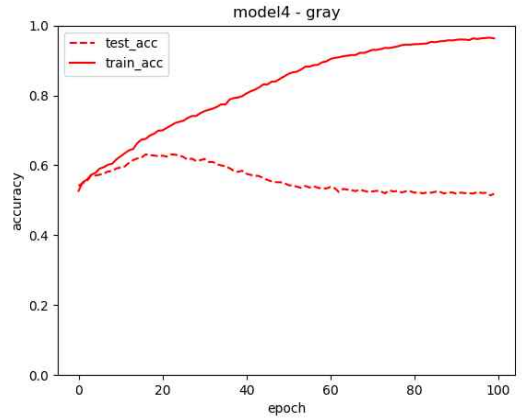

Fig. 14 Performance of Model4 for Preprocessed Image

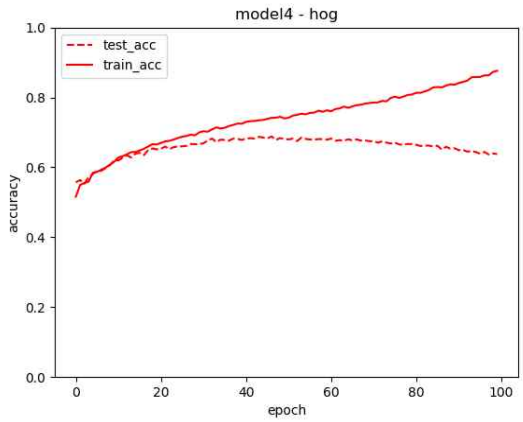

Fig. 15 Performance of Model4 for HOG Image

실험 결과들을 종합해 보면 원 이미지를 사용하여 마지막 모델로 훈련했을 때 테스트 셋에 대한 정확도가 소수 셋째 자리에서 반올림한 82.53%로 가장 높게 나왔다. 모델에 무관하게 특정 시점부터 반복회수를 늘려도 정확도는 높아지지 않는 현상을 보였다. 각 모델의 최고 정확도는 Table 7과 같다. 표에서 보는 바와 같이 DNN 모델을 사용한 경우에는 원 이미지나 전처리 이미지나 호그 이미지나 성능에 큰 차이가 없음을 알 수 있다. 심지어 전처리 이미지인 경우가 원 이미지를 이용했을 경우보다 좋은 성능을 보임을 알 수 있다. 하지만 CNN 모델을 적용하였을 경우는 컨볼루션 층의 깊이에 따라 달라지지만 층이 깊을수록 원 이미지를 이용할 경우의 성능이 훨씬 좋음을 알 수 있다. 호그 특징이 DNN과 결합되었을 때보다 CNN과 결합이 되었을 때 보다 효과가 좋음을 알 수 있다. 즉, 호그 특징과 CNN이 결합되었을 때 그 성능이 전처리 이미지의 성능을 능가함을 알 수 있다. 그리고 이미지의 형태에 상관없이 CNN을 사용할 경우가 DNN을 사용할 경우보다 좋음을 확인할 수 있다. 또한 이미지의 형태에 상관없이 CNN의 층이 깊어지면 깊어질수록 그 성능이 향상됨을 알 수 있다.

Table 7 Accuracies of Models (10 Times Average)

	Model1	Model2	Model3	Model4
RGB	0.507237	0.55625	0.734868	0.825329
Preprocessed	0.535855	0.525987	0.584539	0.645888
HOG	0.527467	0.667105	0.707072	0.701316

훈련하는데 걸린 시간은 Tabel 8과 같다. 모델에 상관없이 이미지 크기 (RGB 이미지 > 전처리 이미지 > 호그 이미지)에 비례하여 시간이 더 소요됨을 알 수 있다. 단, 모델1의 경우는 호그 이미지가 전처리 이미지보다 더 시간이 소요되고 있다. CNN인 경우 층이 깊어지며 깊어질수록 더 시간이 소요될 것 같은데 그렇지 않은 것으로 보인다. 아래 표의 모델3과 모델4를 비교해 보면 원 이미지와 전처리 이미지에 대해서는 모델3이 더 소요됨을 알 수 있다. 이는 모델4에 컨

볼루션층 4개가 추가되었지만 대신에 첫 번째 덴스층의 노드의 수가 32768에서 8192로 1/4 가량으로 줄어들어 총 학습시간이 줄어든 것으로 보인다. 호그 이미지에 대해서는 모델4가 더 소요되었는데 이는 이미지의 크기가 작아 추가된 4층의 컨볼루션층으로 인한 부담이 덴스층 뉴런수 감소에 따른 이득을 초과하는 것으로 보인다.

Table 8 Training Times of Models (10 times average)

	Model1	Model2	Model3	Model4
RGB	397.668	1703.448	1925.114	1804.848
Preprocessed	171.103	1442.591	1658.339	1519.383
HOG	269.366	567.556	546.682	702.824

5. 결 론

본 논문은 기존 전처리 방법과 특징 추출 방법이 딥러닝 모델과 결합되었을 때 어떤 효과를 유발하는지를 검증하기 위해 진행되었다. CNN 구조를 조금씩 변경, 즉 컨볼루션층을 조금씩 늘리면서 실험을 진행한 결과, 일부 모델에서 전처리한 이미지를 사용한 경우가 성능이 좋은 경우도 있었지만 일반적으로 전처리하지 않는 이미지를 사용한 경우가 성능이 훨씬 좋았다. 이는 전처리 과정 중에 정보의 손실이 발생하여 학습 방법이 아무리 좋더라도 성능 향상에 한계가 있음을 의미한다. 호그 특징값으로 구성된 이미지인 경우도 그 결과가 비슷하다. 이는 호그 특징만으로 주어진 학습 이미지를 표현하기에는 부족하고 이 학습 데이터에 맞는 특징들을 사용하여야 함을 의미하며 그러한 특징들은 원 이미지를 이용하여 학습 방법을 통하여 찾아내는 것이 좋음을 뜻한다.

성능적인 측면에서는 전처리 방법과 특징 추출 방법이 부정적인 영향을 미치지만 학습 속도 측면에서는 많은 이득이 있음을 확인할 수 있었다. 성능과 학습 속도 사이에 tradeoff가 존재하기 때문에 지난 수십 년간 진행되어 왔던 전처리 방법과 특징 추출 방법을 효과적으로 활용할 수 있는 새로운 방법을 모색한다면 성능을 조금 희생하면서 속도를 향상시킬 수 있는 방법을 찾을 수 있을 것이다.

References

[1] http://index.go.kr/potal/stts/idxMain/selectPoSttsIdxMainPrint.do?idx_cd=1632&board_cd=INDX_001

[2] Kim, Y. J. and Kim, E.G., “Image Based Fire Detection Using Convolutional Neural Network,” Journal of the Korea Institute of Information and Communication Engineering, Vol. 20, No. 9, pp. 1649-1656, 2016.

[3] T. Celik and H. Demirel., “Fire Detection in Video Sequences Using a Generic Color Model,” Fire Safety Journal, Vol. 44, No. 2, pp. 147-158, 2009.

[4] Ko B. C., Ham, S. J. and Nam, J. Y., “Modeling and Formalization of Fuzzy Finite Automata for Detection of Irregular Fire flames,” IEEE Trans. Circuits Syst. Video Technol., Vol. 21, pp. 1903-1912, 2011.

[5] P. Foggia, A. Saggese, M. Vento, “Real-time Fire Detection for Video-Surveillance Applications Using a Combination of Experts Based on Color, Shape, and Motion,” IEEE Trans. Circuits Syst. Video Technol., Vol. 25, pp. 1545-1556, 2015.

[6] M. Mueller, P. Karasev, I. Kolesov, A. Tannenbaum, “Optical Flow Estimation for Flame Detection in Videos,” IEEE Trans. Image Process., Vol. 22, pp. 2786-2797, 2013.

[7] B.U. Töreyin, Y. Dedeoğlu, U. Güdükbay, A.E. Cetin, “Computer Vision Based Method for Real-Time Fire and Flame Detection,” Pattern Recogn. Lett., Vol. 27, pp. 49-58, 2006.

[8] Luo R.C., Su K.L., "Autonomous Fire-Detection System Using Adaptive Sensory Fusion for Intelligent Security Robot," IEEE/ASME Trans. Mechatron., Vol. 12, pp. 274-281, 2007.
[9] P.V.K. Borges, E. Izquierdo, "A Probabilistic Approach for Vision-Based Fire Detection in Videos," IEEE Trans. Circuits Syst. Video Technol., Vol. 20, pp. 721-731, 2010.
[10] Khan M., Jamil A., and Baik, S.W., "Early Fire Detection Using Convolutional Neural Networks during Surveillance for Effective Disaster Management," Neurocomputing, Vol. 288, pp. 30-42, 2018.
[11] Frizzi S., Kaabi R., Bouchouicha M., Ginoux J. M., Moreau E. and Fnaiech F., "Convolutional Neural Network for Video Fire and Smoke Detection," IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, pp. 877-882, 2016.
[12] J. Sharma, O.-C. Granmo, M. Goodwin, and J. T. Fidje, "Deep Convolutional Neural Networks for Fire Detection in Images," in International Conference on Engineering Applications of Neural Networks, pp. 183 - 193, 2017.
[13] Piao, F. J., Ryu, J. G., Moon, K. S. and Kim, J.N., "A YCbCr Color Model For Fire Detection Based On Fire Movement," In Proceedings of 2010 Korea Multimedia Society Spring Conference, Vol. 13, No. 2, pp. 431-433, 2010.
[14] Park, K. W. Bang, J. S. and Kim, B. M., "Performance Evaluation of Car Model Recognition System Using HOG and Artificial Neural Network," Journal of the Korea Industrial Information Systems Research, Vol. 21, No. 5, pp. 1-10, 2016.
[15] http://srdas.github.io/DLBook/ConvNets.html

이 정 환 (JeongHwan Lee)

• 정회원
• 2014년 ~ 현재 : 국립금오공과대학교 소프트웨어공학과 학부생
• 관심분야 : 인공지능, 영상처리

김 병 만 (Byeong Man Kim)

• 정회원
• 1987년 : 서울대학교 컴퓨터공학과 학사
• 1989년 : 한국과학기술원 전산학과 석사
• 1992년 : 한국과학기술원 전산학과 박사
• 1992년 ~ 현재 : 국립금오공과대학교 교수
• 1998년 ~ 1999년 : 미국 UC, Irvine 대학 방문교수
• 2005년 ~ 2006년 : 미국 콜로라도 주립대학대학 방문교수
• 관심분야 : 인공지능, 정보검색, 정보보안

신 윤 식 (Yoon Sik Shin)

• 정회원
• 1982년 : 경북대학교 학사
• 1985년 : 한국과학기술원 전산학과 석사
• 1988년 ~ 현재 : 국립금오공과대학교 교수
• 관심분야 : 소프트웨어공학, 데이터베이스