공 학 석 사 학 위 논 문

CNN을 이용한 스티어링 휠 이종 불량 검출에 관한 연구

2020년 8월

부경대학교 대학원

정보시스템협동과정

박 광 혁

공 학 석 사 학 위 논 문

CNN을 이용한 스티어링 휠 이종 불량 검출에 관한 연구

지 도 교 수 김 창 수

이 논문을 정보시스템협동과정 석사 학위논문으로 제출함.

2020년 8월

부경대학교 대학원

정보시스템협동과정

박 광 혁

박광혁의 공학석사 학위논문을 인준함.

2020년 8월 일

위 원 장 경영학박사 김 하 균 (인)

위 원 이학박사 이 경 현 (인)

위 원 공학박사 김 창 수 (인)

< 목 차 >

그 림 목 차

표 목 차

A Study on the Detection of Steering Wheel Dissimilarity using CNN

Kwang hyuk Park

Interdisciplinary Program of Information Systems
Pukyong National University

Abstract

Steering wheel that we handle the most driving cars has many switches not only with steering function but with controlling a number of other functions within the cars. Therefore, Steering wheel is equipped with completely different switches even for the same model according to consumers' choices for options. Steering wheel manufacturers produce diverse steering wheels following consumers' needs and then supply those to car manufacturers but very often the human error happens unintentionally, when option "A" needs to be assembled, option "B" comes out as a result, due to various options on the list. Vision Test Machine is used to avoid this human error, but It works simply in the way of comparing images and therefore miscalculations caused by micro conditions occur which are obvious demerits not to mention being inefficiency. In this paper, I present one of the deep learning models that shows outstanding performance in classifying images and assorts car steering wheels using Convolution Neural Network. Steering wheel images collected from 2016 to 2019, made by Vision Test Machine in Steering wheel manufacturer. All collected images are defined as 5 categories. One category of them is used in this paper. Not the whole picture of the image, only 3parts of an image which have an effect on classifying steering wheel are extracted and used. Convolution Neural Network uses a model based Inception module and a model based ResNet. Google's deep learning open source framework, Tensorflow, is used for the experiment and OpenCV and Scikit-Learn are used for preprocessing images and processing data. Steering wheel categorization-model proposed in the paper can be applied in Vision Test Machine System or in actual work adopting additional data processing steps.

Ⅰ. 서 론

1. 연구의 배경과 목적

2차 산업혁명 이후 대부분의 소비재는 소품종 대량생산으로 생산량을 높이는 것에 치중하여 많은 소비자들에게 동일한 제품을 제공하였으나, 현대 사회에서는 획일화된 소비패턴에서 벗어나 개성을 추구하며 다양한 물건을 찾는 소비자들의 욕구를 충족시키는 방향으로 변화되었다. 자동차 산업에서도 이런 흐름에 따라 똑같은 차종이라 하더라도 개개인의 필요와 취향에 따라 다양한 옵션을 선택할 수 있도록 소비자들에게 선택권을 주고 있다. 소비자들이 필요로 하는 사양에 따라 추가되는 기능을 컨트롤하기 위한 스위치와 버튼의 종류 역시 다양해졌다. 대부분의 스위치들은 운전자의 손에 가장 닿기 편한 조향장치, 우리가 편히 핸들이라고 부르는 스티어링 휠에 위치하고 있는 경우가 많다.

하지만 제조업체의 입장에서는, 비슷하게 생겼지만 옵션에 따라 다양한 스티어링 휠을 생산하다보면 사양을 잘못 판단하는 휴먼에러가 발생하기 마련이다. 예를 들면, 스마트 크루즈 기능 옵션을 가진 차량을 선택하여 해당 기능을 포함한 차량을 생산하였지만 이를 작동하는 스위치 대신 블루투스로 연결 된 스마트폰으로 전화 통화를 하는 버튼만이 부착되어 있다면 해당 차량은 스마트 크루즈 기능을 사용 할 수 없다. 이런 제품의 경우에는 스티어링 휠 자체의 성능이나 외관에는 아무 문제가 없다 하더라도 불량으로 처리되어 물류 이동으로 인한 낭비와 UPH[1)]의 하락

을 야기하므로 심각한 손실로 취급하고 있다.

아래의 그림 1과 그림 2 두 제품 모두 H자동차 울산공장에서 생산중인 차량(코드명 : TL)에 장착되는 스티어링 휠이다. 그림 1은 23A라는 사양의 제품이고 그림 2는 24B라는 사양의 제품이다. 얼핏 보기에는 이 두 스티어링 휠이 같은 제품으로 보이지만, 자세히 보면 23A 사양의 우측[2]에 위치한 OK 스위치가 24B에서는 볼륨 컨트롤 버튼으로 들어가 있고, 대신에 그 자리에 크루즈 스위치가 그림으로 표시되어 있다. 그리고 중간부에 댐퍼의 유무에도 차이가 있다. 위에서 예를 들었던 것과 마찬가지로, H자동차에서 23A가 장착되어야 하는 차량이 조립 중인데 24B가 잘못 납품이 되면 불량인 자동차가 생산되는 것이다. 조립이 되기 전에 잘못 납품 된 부품을 찾아낸다 하더라도 컨베이어 벨트 시스템으로 생산중인 자동차 공장 전체가 멈추기 때문에 큰 손실을 끼칠 뿐 아니라, 재납품을 통한 물류비용의 추가 발생과 납품 된 제품의 반출이 어려워 발생하는 손실비용까지 연쇄적인 비용발생으로 인해 막대한 피해가 발생한다.

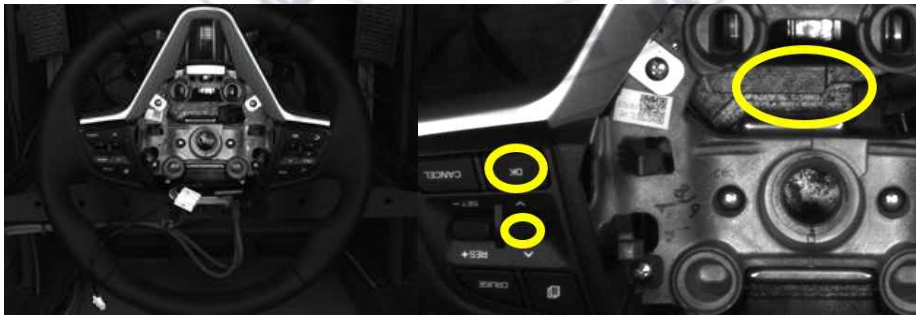


그림 1. 스티어링 휠(TL - 23A)의 사진

---

1) Unit Per Hour. 시간 당 생산량.
2) 비전 테스트기에서 생성되는 이미지가 뒤집혀 있더라도, 스티어링 휠이 장착되어 정 위치에 있는 것을 기준으로 오른쪽과 왼쪽을 정의한다.

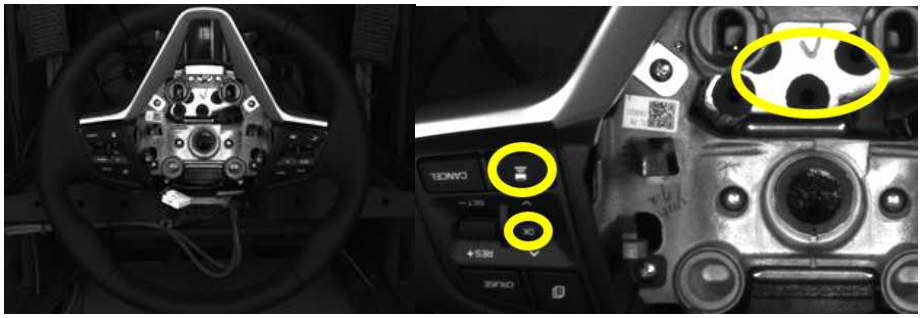


그림 2. 스티어링 휠(TL - 24B)의 사진

이런 손실비용의 발생을 방지하고 보완하기 위하여 인원을 투입하여 직접 검사를 하는 1차원적인 방법을 도입하여 운영 중이지만 역시나 휴먼에러가 발생할 확률이 높아 불량 유출을 막는 궁극적인 방법이 될 수 없고 비용이 많이 들어가는 단점이 있다. 그리하여 2차적으로 제품을 다각도에서 촬영하여 검사하는 비전 테스트기도 도입하여 운영 중이지만, 기존의 비전 테스트기는 OpenCV를 활용한 템플릿 매칭으로 단순 이미지 비교를 통한 판별을 하고 있기 때문에 고정 시키는 부위가 틀어지거나, 조명의 변화, 제품의 마감 상태 등 검사 조건의 미세한 변화에도 정상적인 판별을 할 수 없어 어려움을 겪고 있다.

본 논문은 자동차 내장 부품 회사에서 현재 쓰이고 있는 비전 테스트기의 한계를 극복하기 위한 방법으로 가장 눈에 띄는 부품인 스티어링 휠의 데이터의 분석에 딥러닝[5]을 도입하여 생산성과 효율성을 높이는 방법을 제안하고 이를 적용하여 구현하여 성능을 평가해 보는 것이 목적이다.

2. 연구 범위

본 연구에서는 스티어링 휠의 이종 불량 검출을 위한 이미지 비교 분석을 단순 이미지 비교가 아닌 머신러닝을 통한 학습이 가능하도록 이론적 배경인 합성곱 신경망과 합성곱 신경망(Convolutional Neural Network, CNN) 기반의 아키텍처들에 대해 고찰하고, CNN을 기반으로 하는 스티어링 휠 분류 모델을 설정하여 실제 데이터를 학습시켜보았다.

본 연구는 전체 5장으로 구성되어 있으며 각 장의 구성은 다음과 같이 정리하였다. 제1장 서론 부분은 이 연구를 하게 된 배경과 연구의 목적을 서술하고, 연구의 범위에 대해 소개하였다. 제2장은 머신러닝을 기반으로 하는 스티어링 휠 분류에 필요한 합성곱 신경망과 CNN을 기반으로 하는 여러 아키텍처의 종류들을 알아보고 머신러닝을 기반으로 하는 부품 분류의 범주에 따른 차이를 알아보았다. 제3장에서는 앞서 알아본 이론적 배경들과 실제 데이터들을 기반으로 하여 분류한 스티어링 휠의 이미지를 CNN을 통해 이진 분류와 다중 분류 모델로 학습시켜보았다. 제4장에서는 3장의 분류를 통해 수집된 데이터의 전처리와 모델학습을 해보았고, 그 성능을 평가해보았다. 제5장은 연구 결과를 요약하고, 연구 결과가 실무에서 가지는 의미에 대한 정리와 함께 본 연구가 가지고 있는 한계점에 대해 파악하고 향후 연구의 방향에 대해서도 제시하였다.

Ⅱ. 이론적 배경 및 선행연구

1. 합성곱 신경망

합성곱 신경망(Convolutional Neural Network, CNN)은 1998년 얀 르쿤(Yann LeCun)교수의 논문[6]에서 손글씨 숫자 데이터의 분류를 하는 것을 시작으로 소개 되었으며, 이미지 등과 같이 격자 형태로 배열된 데이터를 처리하는데 특화된 신경망이다. 합성곱 신경망은 기존의 신경망, 즉 순방향 다중 퍼셉트론(feed-forward multi perceptron)과 다르게 완전연결계층(fully connected layer)이 아니라 합성곱층(convolutional layer)과 풀링층(pooling layer)으로 구성된 것이 특징이다.

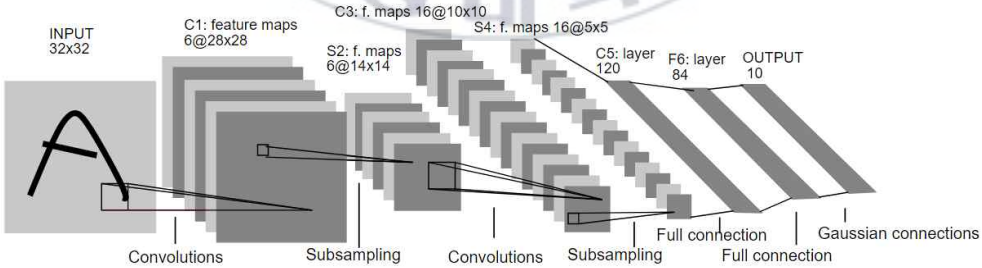


그림 3. 합성곱 신경망의 최초 모델:LeNet

가. 합성곱층

합성곱층은 합성곱 신경망의 주요 연산에 대한 레이어를 의미하며 필터(커널)를 사용해 주어진 데이터에 합성곱 연산을 진행한다. 전통적인 신경망의 완전연결 계층은 매개변수들의 전체에 대한 행렬 곱을 수행한다. 결과적으로 모든 출력 단위는 모든 입력단위의 영향을 받는다. 이와 다르게 합성곱은 필터에 대한 국소적 연산을 진행한다. 이는 합성곱층이 희소 상호작용(Sparse Interaction)이라는 성질을 갖게 만들고 기존의 완전연결 계층 보다 상대적으로 더 적은수의 가중치를 가지게 하고, 메모리 사용량을 줄어들게 하고, 출력을 위한 연산의 수 또한 줄어들게 한다. 합성곱층의 존재 여부에 따라 합성곱 신경망의 구분여부가 판단된다.

나. 풀링층

풀링층은 특정 위치에서의 신경망의 출력을 근처 출력들의 요약 통계량(summary statistic)으로 대체한다. 통계량을 추출하는 방법은 최댓값 또는 평균값을 계산하는 방법이 존재한다. 어떤 경우이더라도 풀링은 표현이 입력의 작은 이동에 대해 근사적으로 불변이 되게 하는데 도움을 준다. 이는 어떤 특징이 구체적인 위치가 아니라 그런 특징의 존재 여부 자체에 대해 탐지하는 경우에 유용하다. 즉, 이동에 대한 불변성을 탐지하는데 유용하다.

2. 합성곱 신경망 기반의 아키텍처

합성곱 신경망은 LeNet 이후 많은 발전이 있었다. 특히 ImageNet의 ILSVRC라는 이미지 분류, 이미지 분할, 객체 탐지에 대한 이미지 처리 대회를 기반으로 매년 새로운 합성곱 신경망 아키텍처가 이 대회에서 우승하였고 우승한 모델은 이미지 분류의 표준 합성곱 신경망 모델이 되었다. 아래 그림 4에서 볼 수 있듯이, 2015년의 우승 모델인 ResNet은 훨씬 더 깊은 레이어를 가지고 있고, 초기 모델에 비해 월등히 낮은 에러율을 보여주고 있다. 합성곱 신경망 기반의 아키텍처는 해를 거듭할수록 뛰어난 성능의 모델이 나오고 있다.

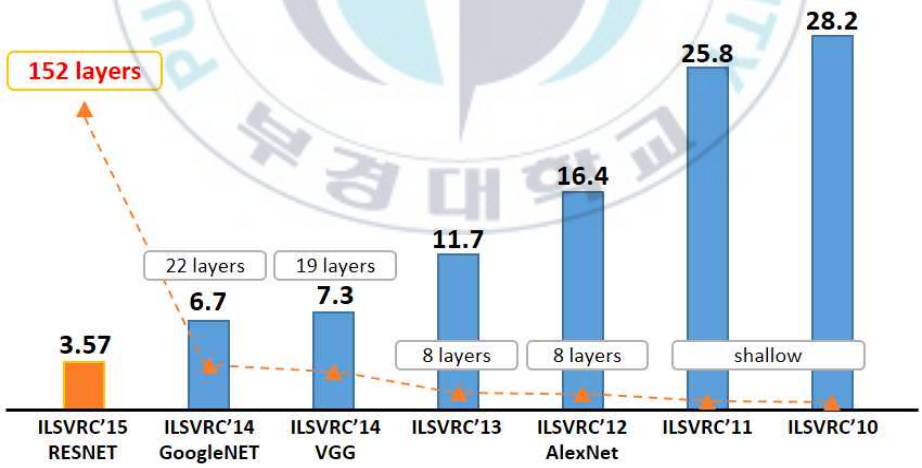


그림 4. 연도별 ILSVRC 우승 모델 구조의 레이어와 에러율

가. AlexNet

AlexNet[7]은 딥러닝 시대의 초기 CNN모델이다. ILSVRC이라는 이미지 분류 대회의 2012년도 우승모델이다. 본격적인 딥러닝의 시작은 2012년에 AlexNet부터라고 볼 수 있다.

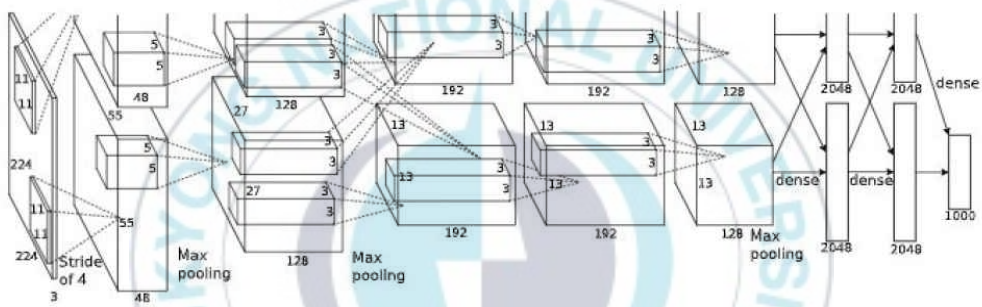


그림 5. AlexNet의 아키텍처

총 5개의 합성곱층과 3개의 완전연결계층으로 구성되어 있고 당시 메모리 사용의 한계로 인해 2개의 병렬구조로 구성되었다. AlexNet은 이전과 다르게 활성화함수를 sigmoid나 tanh가 아닌 ReLU를 사용했으며 특징맵 정규화를 위해 Local Response Normalization(LRN)를 사용했다. 또한 과적합을 피하기 위해 Dropout계층이 포함되어있다.

나. VGG

VGGNet[8]은 옥스포드 대학의 Visual Geometry Group연구소에서 개발한 합성곱 신경망 모델로 ILSVRC 2014년도 우승 모델이다. AlexNet 다음세대의 합성곱 신경망 모델이라고 할 수 있으며 논문의 이름에서 알 수 있듯이 보다 많은 합성곱 레이어로 구성된 딥러닝 아키텍처 이다. 종류에 따라 VGG11, VGG13, VGG16, VGG19로 구분되며 모델이름의 숫자는 모델에서 학습가능한 계층의 개수를 의미한다.

표 1. VGG 모델 비교

ConvNet 구성						
종류	A	A-LRN	B	C	D	E
학습 가능한 레이어 개수	11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
입력 데이터	입력 (224*224 RGB image)					
합성곱	conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
풀링	maxpool					
합성곱	conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
풀링	maxpool					
합성곱	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**

풀링	maxpool					
합성곱	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
풀링	maxpool					
합성곱	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
풀링	maxpool					
완전 연결	FC-4096					
	FC-4096					
	FC-1000					
출력	soft-max					

표 1은 VGG 논문에 제시된 여러 가지 VGG 아키텍처를 나타낸다. A는 11개의 학습 가능한 레이어가 포함된 합성곱 신경망을 나타내고 B는 13개, C와 D는 16개 E는 19개의 학습 가능한 레이어를 포함한다. 여기서 학습 가능하다는 것은 풀링 레이어와 소프트맥스를 제외한 역전파로 학습할 가중치를 가지고 있는 합성곱 레이어와 완전연결 레이어를 의미한다. A-LRN의 LRN층은 Local Response Normalization층을 의미하며 이 레이어는 신경망의 구조가 깊어졌을 때 발생하는 경사소실을 막아주는 정규화를 수행한다. VGG에서는 A와 A-LRN의 실험결과를 제시함으로써 Local Response Normalization층이 성능개선에 무의미하다는 것이 제시되었다. C와 D는 16개의 같은 수의 학습 가능한 레이어로 구성되어 있지만 합성곱 레이어의 필터 사이즈가 1x1과 3x3 적용에 대한 차이가 존재한다.

다. GoogLeNet

GoogLeNet[9]은 이름에서 알 수 있듯이 구글에서 개발한 CNN기반 모델로 Inception 모듈을 사용해 아주 깊은 신경망 구성을 가능하게 한 모델이다. 아래의 그림 6에서 볼 수 있듯이 파란색은 Convolution, 빨간색은 Pooling, 노란색은 Softmax, 초록색은 Concat/Normalize를 나타낸다. 신경망은 깊어질수록 반복 곱셈 연산으로 인해 정보가 점점 손실되는 단점을 가지고 있다. 또한 깊이가 깊어질수록 학습해야하는 가중치가 많아지는데 이를 Inception 모듈을 사용해 해결하였다. Inception 모듈을 사용한 덕에 엄청나게 깊은 층을 가지게 되지만, 1x1 Conv를 자주 사용하여 연산의 총량을 줄일 수 있게 하였다. 그림 7에서 간단하게 표현하였듯이 그리고 중간에 Pooling 레이어를 추가로 삽입하여 크기를 줄일 수 있게 해둔 것과 Softmax를 통해 결과를 뽑아내는 부분 역시 중간에 있다는 것이 특징이다.

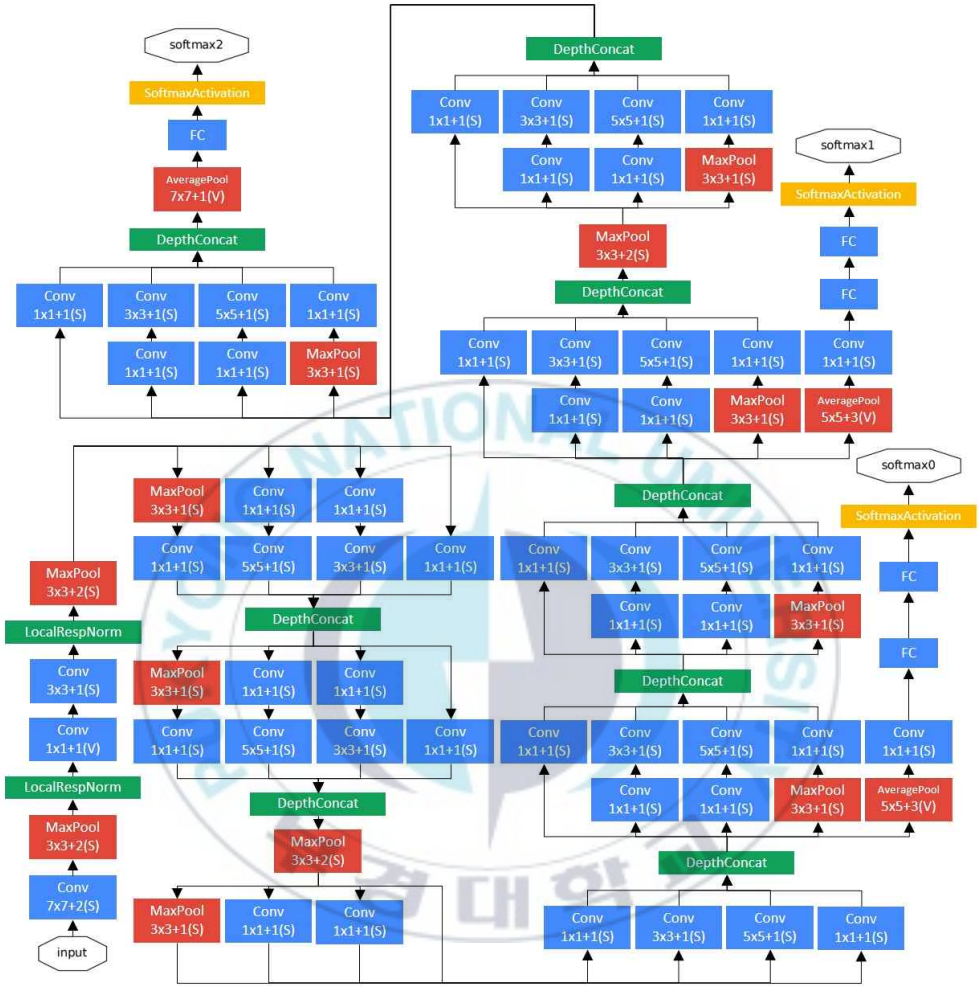


그림 6. GoogLeNet 모델 구조

그림 6은 GoogLeNet의 모델 구조를 나타낸다. GoogLeNet은 기존 모델들과 다른 특징이 있는데 첫 번째 특징은 여러 개의 아웃풋을 가지는 것이고 두 번째 특징은 인셉션 모듈이 사용된 것이다. 입력으로부터 초

기에는 합성곱 레이어와 풀링 레이어 그리고 Local Response Normalization 레이어로 구성된 구조가 두 번 사용되고 이후 인셉션 모듈 9개로 구성된다. 먼저 여러 개의 아웃풋을 가지는 특징을 살펴보면 기존의 신경망을 사용할 때 깊이가 깊어지면 발생하는 경사손실 문제를 네트워크 구조의 중간에 출력을 만드는 것으로 해결했다. 두 번째 특징인 인셉션 모듈은 여러 개의 합성곱 레이어를 통해 학습할 파라미터 수를 효율적으로 낮춘다. 이는 인셉션 모듈 내에 포함된 1x1 합성곱을 통해 수행되는데 1x1 합성곱은 채널을 줄여주기 때문이다. 그림7은 하나의 인셉션 모듈을 의미한다.

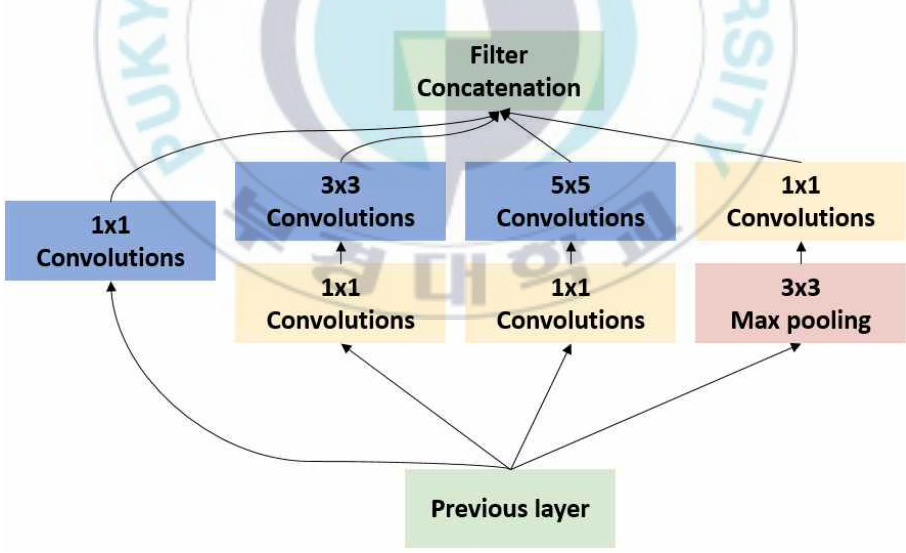
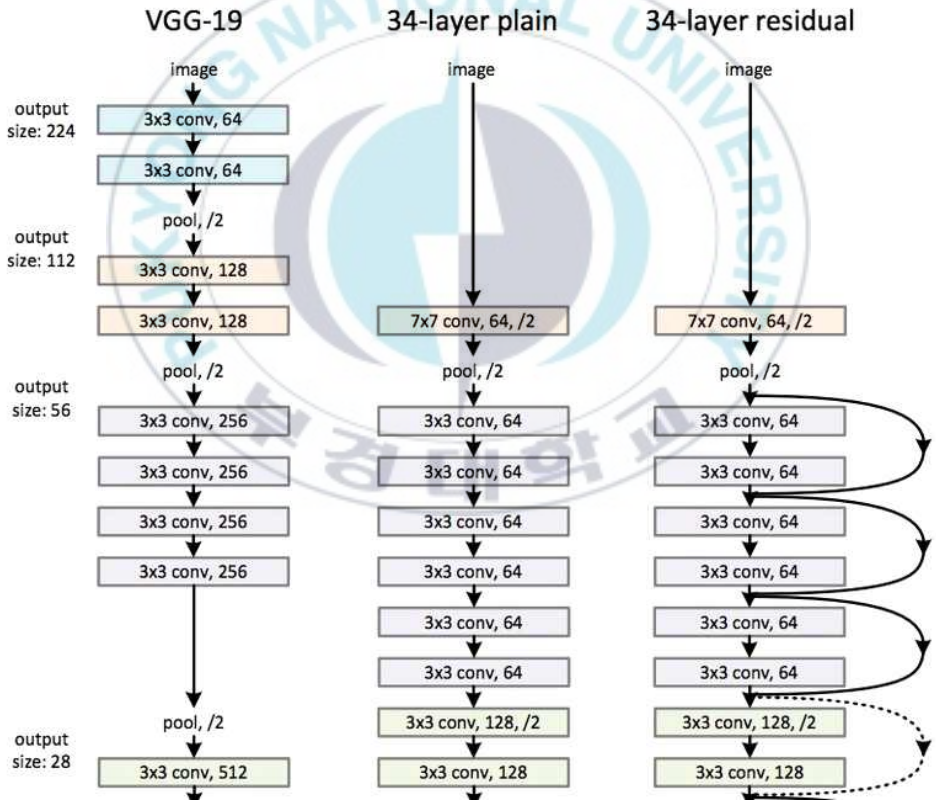


그림 7. Inception 모듈의 구조

라. ResNet

GoogLeNet에서 언급한 것과 같이 신경망의 깊이가 깊어지면 발생하는 경사 손실 문제를 잔차연결[3], 즉 아무 연산을 하지 않고 깊은 레이어로 정보를 전파하면서 동작하는 모델이 ResNet[10]이다. 마이크로소프트의 북경 연구소에서 개발한 알고리즘으로, 초기 152레이어로 구성했으며 1001레이어까지 확장을 했다.



3) 잔차연결은 하위 층의 출력 텐서를 상위 층의 출력 텐서에 더해서 아래층의 표현이 네트워크 위쪽으로 흘러갈 수 있도록 한다. 즉, 하위 층에서 학습된 정보가 데이터 처리 과정에서 손실되는 것을 방지한다.

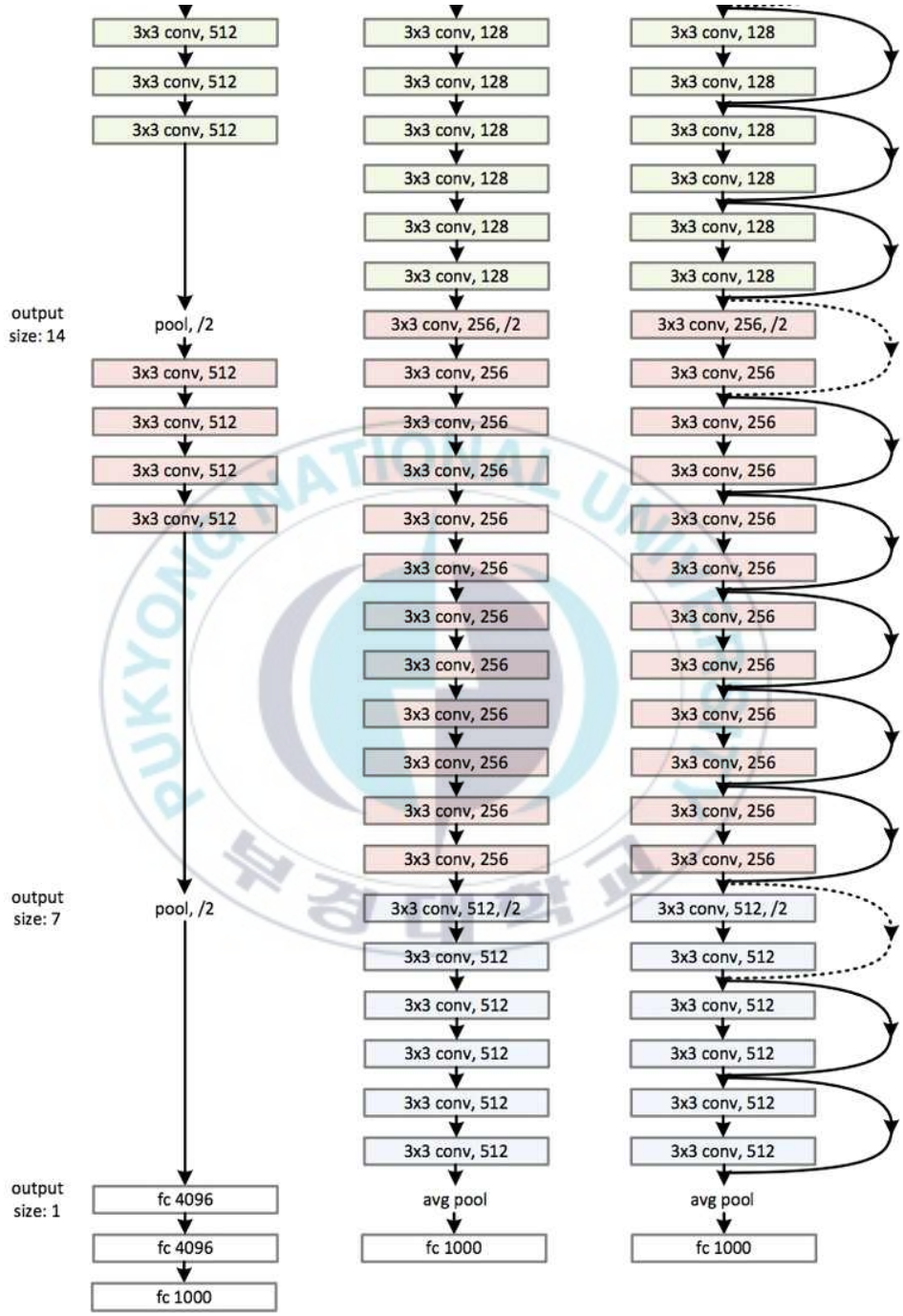

3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
output size: 14
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
output size: 7
pool, /2
output size: 1
fc 4096
fc 4096
fc 1000
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000

그림 8. ResNet

사람이 이미지를 분류하면서 발생하는 에러율을 5%라고 보는데, ResNet은 그림 4에서 나타냈듯이, 3.6%도 되지 않는 에러율을 보이면서 최초로 사람을 뛰어넘은 학습 모델이 되었다.

그림 8은 VGG와 신경망, ResNet을 비교한 그림이다. 기존의 구조와 다르게 ResNet은 합성곱 레이어를 건너뛰는 연결들이 존재한다. 이 연결이 잔차 연결이며 이전 레이어의 입력을 합성곱 레이어를 통과한 결과와 합쳐 다음 층으로 보낸다. 그림 9는 두 가지의 잔차 연결에 대한 모듈을 보인다. 첫 번째 모듈은 일반적인 잔차 연결이고 두 번째 모듈은 1x1 합성곱을 추가해 GoogLeNet에서와 같이 학습할 파라미터수를 줄인 모듈이다.

그림 9. 잔차 모듈

3. 머신러닝 기반 부품 분류

머신러닝의 종류는 사람의 개입 정도와 데이터 가용성에 따라 여러 가지로 나뉜다. 사람의 개입에 따른 범주화에는 지도 학습과 비지도 학습이 있으며, 데이터 가용성에 따른 학습 방법으로는 배치 학습과 온라인 학습, 엔드 투 엔드 시스템이 있다.

가. 사람의 개입에 따른 범주화

(1) 지도 학습

지도 학습[11]은 학습할 데이터가 (x, y)로 구성되어 있고 입력과 출력 사이의 연관성을 맵핑하여 학습한다. 즉, 문제를 내고 정답까지 함께 알려주는 방식이다. 이것은 사람의 개입이 많이 들어간다는 것을 의미한다. 분류와 회귀 등의 모델이 지도학습에 해당된다. 본 연구의 목적인 스티어링 휠 이종 불량 분류에서는 학습 시 입력 이미지에 해당되는 스티어링 휠의 종류가 이미지별로 레이블링 되어 있어야한다는 것이다.

(2) 비지도 학습

비지도 학습[12]은 입력과 출력의 연관이 아닌 입력 데이터에 내재된 잠재적 구조와 패턴 그리고 관계를 학습한다. 군집화, 차원 축소, 연관 규칙 마이닝 등이 해당된다. 비지도 학습은 부품의 이종 불량이 아닌 부

품을 정상과 비정상으로 분류하는 등의 학습시킨 데이터와 다른 데이터의 입력 여부를 결정하는 모델에 사용될 수 있다.

나. 데이터 가용성에 따른 학습방법

부품 등의 분류를 위한 머신러닝 모델에서 학습을 위한 시스템 또한 중요한 연구 분야이다. 즉 현장에서 새로 생성되는 데이터에 대한 분류를 위해 적용되는 처리방식, 여기서는 학습방식에 대해 기술한다.

(1) 배치 학습 (오프라인 학습)

배치학습[13]은 학습이 일괄적으로 수행된 후 배포된다. 즉 정해진 주기에 따라 기존 데이터와 함께 새로 추가되는 데이터를 같이 다시 학습하는 것을 의미한다. 가장 많이 사용되는 학습방식을 의미한다.

(2) 온라인 학습

온라인 학습[13]은 데이터가 미니 배치로 시스템에 입력되어 계속해서 학습을 진행하는 방식이다.

(3) 엔드 투 엔드 시스템

머신러닝 시스템을 구성하기에 있어서 배치나 온라인 학습과정은 데이터를 준비하고 모델을 학습하고 모델을 배포하는 과정과 함께 구성된다. 최근에는 DevOps[14]와 함께 머신러닝 모델 개발과 운용을 end-to-end

로 하는 방식을 MLOps[15,16]라고 표현한다. 그림 10은 대표적인 머신러닝 모델 운용환경을 의미한다. 모델을 평가하고 부족한 부분을 튜닝한 다음에 모델의 성능이 향상될 때까지 계속 반복을 한다는 것이 가장 중요한 부분이다.

그림 10. end-to-end 시스템

Ⅲ. CNN 기반의 스티어링 휠 이종 분류 모델

이번 장에서는 본 연구에서 제안하는 CNN기반 스티어링 휠 분류 모델을 정의하고 모델에 대한 데이터와 개발 단계를 기술한다. 이를 위해 스티어링 휠 생산 공장에 설치된 비전 테스트기에서 2016년부터 2019년까지 4년 간 수집한 데이터를 사용한다. 스티어링 휠 데이터는 이미지 데이터 이며 3채널의 RGB 색상 세트와 1600px X 1200px 크기를 가진다.

1. 스티어링 휠 데이터

스티어링 휠 이미지 데이터는 총 5가지의 각도에 대한 이미지 데이터이다. 아래 그림 11에서 볼 수 있듯이, 좌측과 우측에 대한 2개씩의 이미지와 전체 이미지가 포함된다.

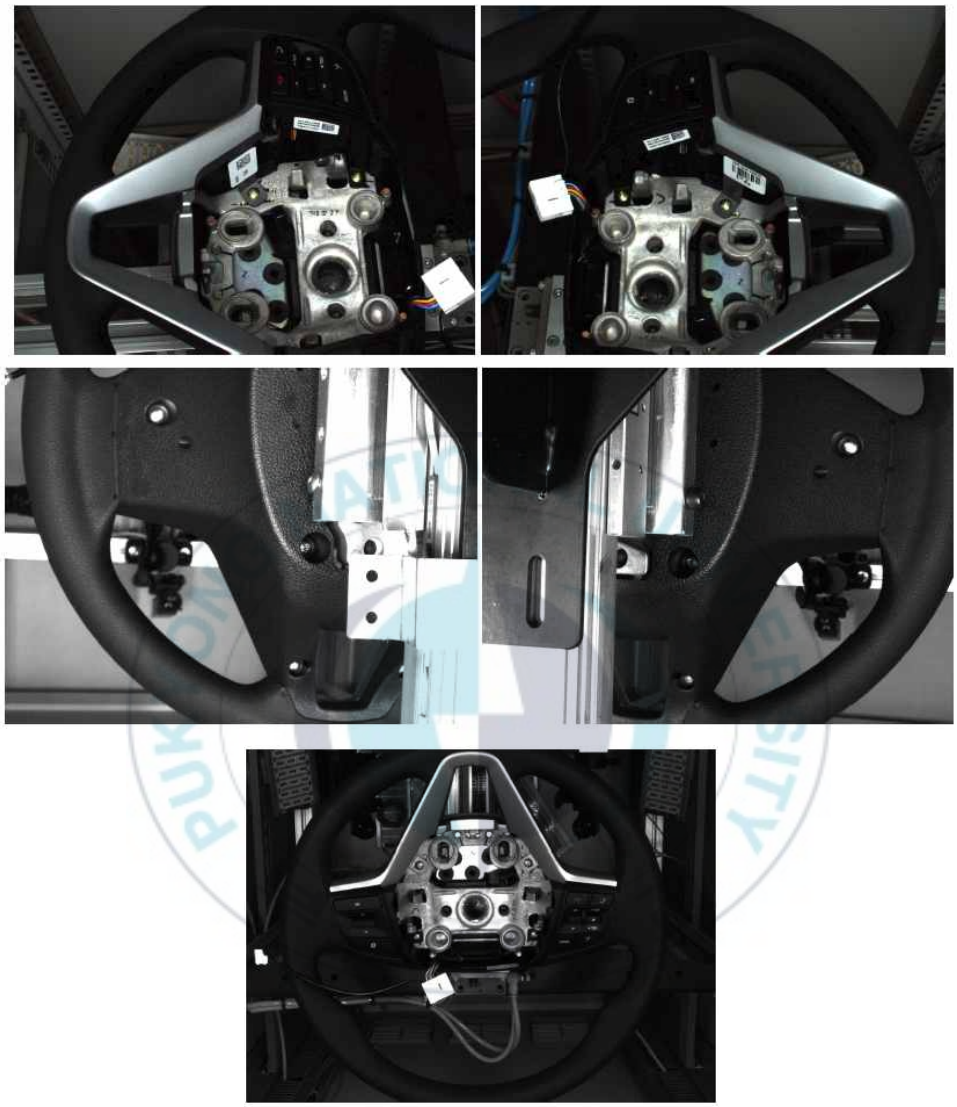


그림 11. 비전 테스트기에서 수집된 스티어링 휠 이미지

표 2. 스티어링 휠 종류 및 종류별 이미지 개수

코드	개수	코드	개수	코드	개수	코드	개수	코드	개수
R70	126,068	01B	2,266	02G	486	24A	88	25U	6
R57	81,491	ADT	2,262	09A	479	P35	76	BFW	6
03A	68,860	11T	2,193	P40	478	R31	60	U43	5
R93	58,048	15A	2,057	R56	470	65B	58	R34	5
P93	44,799	13B	2,044	05B	465	17B	57	66B	5
03F	31,176	25F	1,987	R75	443	S98	48	18A	5
R55	30,348	P73	1,982	13F	400	10B	48	Y60	4
R40	22,231	P53	1,961	AFT	380	06A	45	12T	4
23A	20,476	20B	1,950	Y96	377	P81	41	BGT	4
R53	19,164	21A	1,947	ABT	336	07B	40	17T	4
25A	13,838	00B	1,850	67E	290	U87	37	19B	4
13A	12,015	R77	1,741	67B	279	BDT	36	29B	4
R54	11,914	R50	1,558	ACT	272	65A	29	CFU	3
25B	10,838	P00	1,451	05A	243	65E	29	P91	3
21B	9,879	P30	1,354	24B	229	15F	25	06B	3
P92	8,843	D78	1,334	23C	225	U41	22	BFU	2
01A	7,134	25E	1,298	11B	221	R91	16	43A	2
R78	7,064	R35	1,295	67A	215	P50	16	18B	2
R73	6,424	P54	1,256	P75	202	65C	16	29A	2
R30	5,762	13T	1,248	19A	197	04B	13	66A	2
P55	5,443	R71	1,119	R74	186	17A	13	16A	2
10A	5,231	25C	921	02A	179	TLT	12	THT	1
00A	5,220	R92	887	P10	174	64B	12	PE2	1
68A	5,079	R58	808	U88	171	64E	11	DFT	1
R81	4,975	15B	722	67C	165	64C	10	13U	1
11A	4,811	U86	720	R60	138	TPT	9	15U	1
23F	4,572	R10	697	BFT	138	PE3	9	43B	1
P60	4,570	12B	683	ADU	134	P90	8	10C	1
P56	4,328	AAT	630	03G	120	R80	8		
23B	3,247	C93	629	11U	118	DFU	8		
P57	2,969	20A	610	07A	107	19T	8		
12A	2,804	10T	541	ABU	95	66E	8		
R59	2,457	03B	523	15T	93	64A	7		

(개수 = 실제 데이터의 수 / 5)

표 2는 수집된 스티어링 휠의 분류코드와 데이터개수를 의미한다. 여기서 데이터란 하나의 생산된 스티어링 휠에 대한 데이터를 의미하며 하나의 스티어링 휠 당 5개를 이미지를 수집한다.

2. 분류 범위 정의

CNN은 앞에서 살펴본 것과 같이 이미지 기반의 데이터를 분류하거나 인식하는 작업에 적합한 신경망 구조이다. n x n의 크기를 가지는 여러 개의 커널이 이미지를 순회하며 합성곱 연산을 진행하여 이미지처럼 변환, 스케일링, 회전등이 포함되는 공간 불변 특성을 추론하는데 적합하다. 그렇지만 스티어링 휠처럼 모두 원형의 형태를 가지고 클래스 사이의 아주 작은 차이가 특징을 가지는 경우 많은 특성 엔지니어링을 필요로 한다. 다음은 4개의 각각 다른 분류의 스티어링 휠 이미지이다.

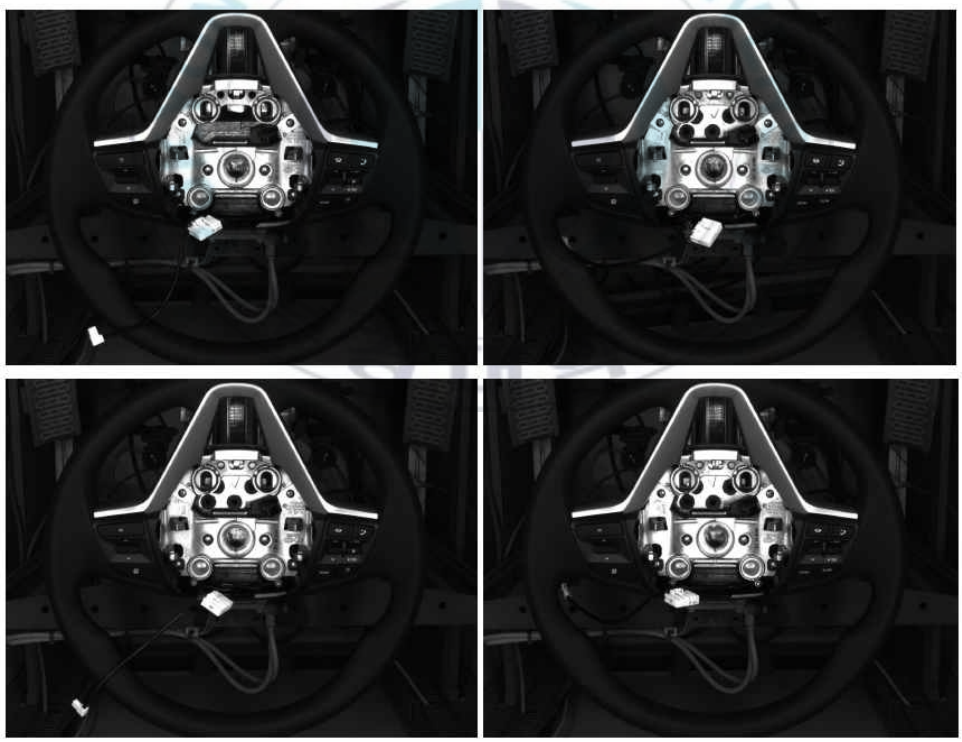


그림 12. 각기 다른 사양의 스티어링 휠 이미지

위 이미지는 실제 다른 분류로 포함되는 스티어링 휠이지만 자세하고 꼼꼼하게 살펴보지 않는다면 육안으로는 쉽사리 그 차이점을 알 수 없다. 때로는 사진에서 보이지 않거나 보이더라도 구분이 불가능한 요소나 특징으로 분류하는 경우도 있다. 이로 인해 여기서는 먼저 스티어링 휠의 분류 범위를 정의한다.

분류 범위는 총 5개의 자동차 스티어링 휠 그룹으로 정의된다. 다음은 분류를 위해 정의된 5개의 분류와 해당 분류에 포함된 실제 분류이다. 각 사양을 구분하고 있는 코드의 특징으로 분류하였고, 각 클래스 별로 스위치의 종류나 댐퍼의 유무 등의 차이가 있다.

표 3. 분류에 따른 데이터 개수

스티어링 휠 분류	데이터 개수
A	779,070
B	180,333
C	11,255
E	10,668
R	1,939,586

3. 특성 엔지니어링

모델 개발을 위해 분류된 5가지 분류의 스티어링 휠을 CNN으로 학습하기 위해 비전 테스트기에서 수집된 이미지를 그대로 사용하지 않고 먼저 특성 엔지니어링을 진행한다. 이는 이미지 전체적으로는 비슷한 형태를 가지고 있으나 각 사양 별 차이점을 가지게 만드는 요소는 이미지의 일부 특정 부분이기에 정확한 분류를 위하여 결정적인 요소를 추려내었다.

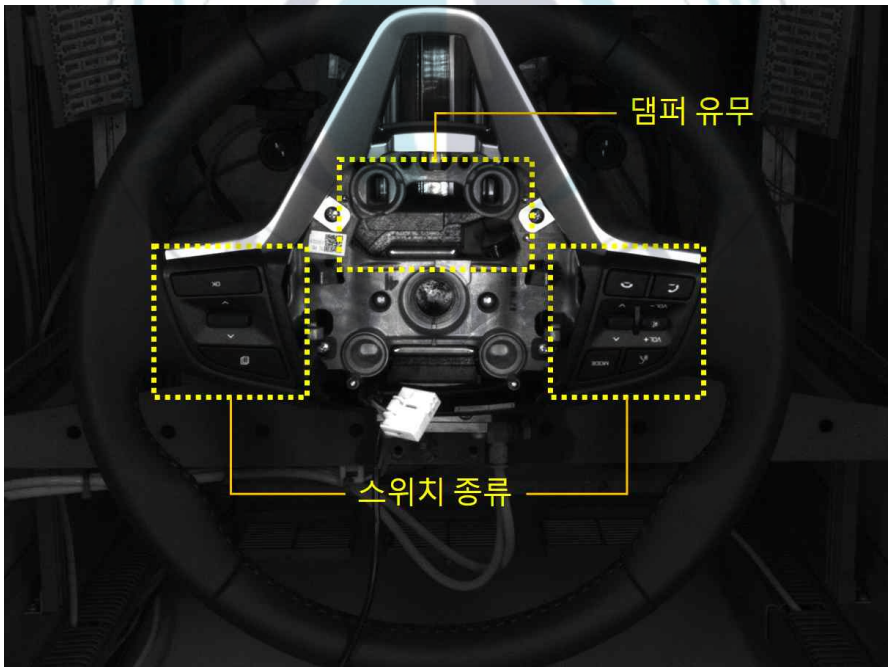


그림 13. 스티어링 휠에서 스위치와 댐퍼

본 연구에서는 스티어링 휠 이미지 전체 부분 중 스위치의 종류 와 댐퍼 유무를 분류 모델의 학습데이터로 사용한다. 전체 이미지 사진에서 해당 부분을 추출하기 위해 템플릿 매칭을 사용한다. 템플릿 매칭이란 원본 이미지가 존재하고 원본 이미지에서 매칭 할 템플릿을 픽셀 간 비교해 템플릿과 가장 유사한 부분을 원본이미지에서 찾는 방법이다. 이때 픽셀 간 비교를 위해 다음과 같은 방법들이 존재한다.

표 4. 템플릿 매칭 종류[14]

템플릿 매칭 종류	수식 표현
Squared Difference	$R(x,y)=\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2$
Squared Difference Normed	$R(x,y)=\frac{\Sigma_{x',y'}(T(x',y')-I(x+x',y+y'))^2}{\sqrt{\Sigma_{x',y'}T(x',y')^2 \cdot \Sigma_{x',y'}I(x+x',y+y')^2}}$
Correlation	$R(x,y)=\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))$
Correlation Normed	$R(x,y)=\frac{\Sigma_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\Sigma_{x',y'}T(x',y')^2 \cdot \Sigma_{x',y'}I(x+x',y+y')^2}}$
Correlation Coefficient	$R(x,y)=\sum_{x',y'}(T'(x',y') \cdot I(x+x',y+y'))$ $T'(x',y')=T(x',y')-\frac{1}{(w \cdot h)} \cdot \sum_{x'',y''}T(x'',y'')$ $I'(x+x',y+y)=I(x+x',y+y)-\frac{1}{(w \cdot h)} \cdot \Sigma_{x'',y''}I(x+x'',y+y'')$
Correlation Coefficient Normed	$R(x,y)=\frac{\Sigma_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\Sigma_{x',y'}T(x',y')^2 \cdot \Sigma_{x',y'}I'(x+x',y+y')^2}}$

Squared Difference는 각 픽셀의 차이의 제곱을 의미하며 원본과 템플릿이 완벽히 일치하면 0이 되고 유사하지 않으면 보다 큰 양수를 가진다. Correlation은 상관관계를 의미하며 원본과 템플릿이 유사하면 보다 큰 양수를 가지고 그렇지 않으면 작은 값을 가진다. Correlation Coefficient는 상관계수를 통해 매칭하는 방법이며 원본과 템플릿의 영상 밝기를 평균 밝기로 보정한 후 상관관계를 계산하는 방식이다. 원본과 템플릿이 유사하면 보다 큰 수를 가지고 그렇지 않으면 0과 유사한 값을 가진다. 템플릿 매칭은 이미지의 크기에 대한 스케일링을 따로 지원하지 않는 방법이다. 즉 정규화에 대한 방법은 따로 포함이 안 된다. 본 연구에서는 스티어링 휠에 존재하는 스위치와 댐퍼 부분을 결정하기 위해 사용되므로 스티어링 휠이 원본 이미지에 해당하고 스위치, 댐퍼 이미지가 템플릿으로 적용된다. 즉 템플릿이 원본 이미지 보다 작다. 다음은 각 부분의 템플릿 이미지와 원본 이미지와의 템플릿 매칭에 대한 유사도 맵, 그리고 원본 이미지에 매칭 된 템플릿의 결과를 의미한다. 먼저 원본 스티어링 휠 전면부 이미지를 살펴보면 다음과 같다. 제품의 특성 상 흑백 이미지로 보이지만, 위에서 언급한 것처럼 RGB값을 가지고 있다. 주로 검은색 가죽으로 구성 된 바디 부분과 좌/우 스위치 부분, 대부분 은색을 띄고 있는 하단 베젤 부분과 알루미늄-마그네슘 합금으로 이루어져 있는 아마추어 코어(Armature Core) 부분이 중앙에 보인다.

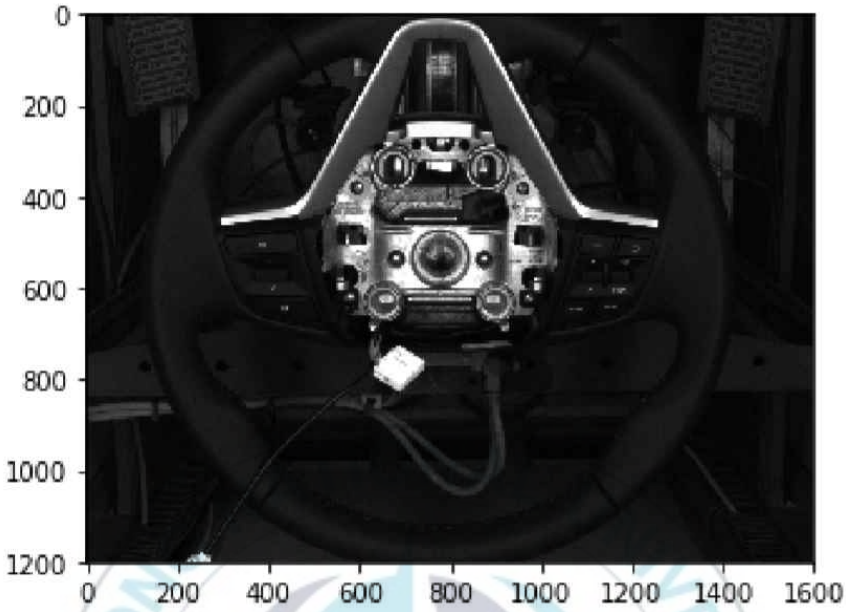


그림 14. 스티어링 휠 전면부 이미지

아래의 그림 15는 위 이미지에서 우측 스위치를 탐지하기 위한 템플릿 이미지이다. 본 이미지는 수작업으로 위 데이터에서 추출한다. 스위치에 위치한 버튼의 개수가 사양마다 다르기 때문에 내부 형태 구조가 다른 경우도 있고, 모양은 같으나 흰색으로 프린팅 된 영문 또는 그림이 다른 경우도 있다.

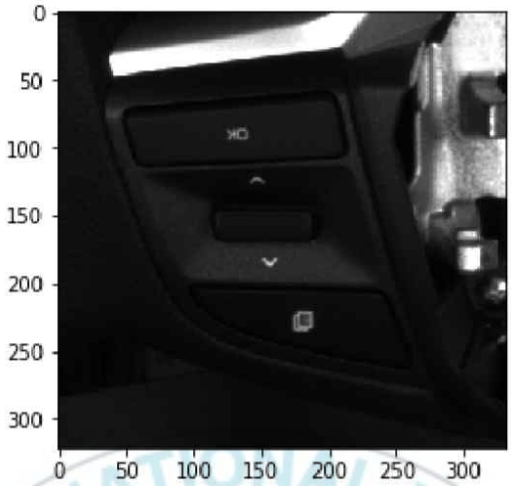


그림 15. 우측 스위치 템플릿 이미지

다음 그림 16은 그림 14 이미지에 그림 15 이미지를 템플릿 매칭 한 결과이다. 매칭을 위해 Correlation을 사용했다.

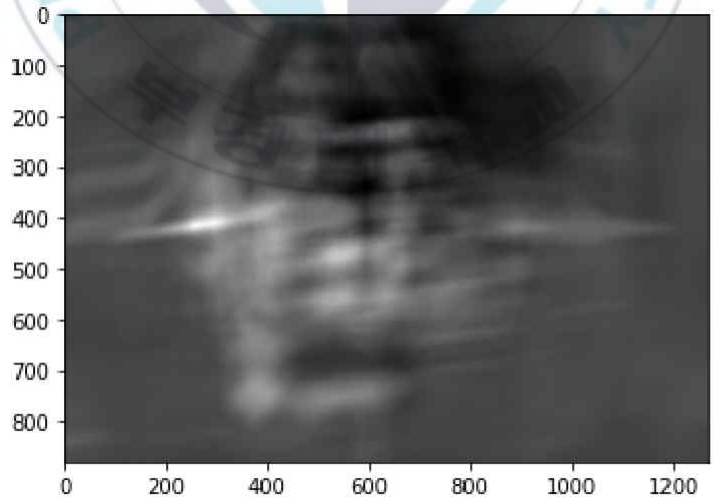


그림 16. 우측 스위치 템플릿 매칭 결과

Correlation의 경우 유사한 부분의 상관관계를 기반으로 매칭이 진행된다. 상관관계에서는 결과 값이 크면 클수록 유사도가 높다고 판단된다. 매칭 결과 이미지를 보면 가장 밝은 점이 존재하는데 이미지의 경우 값이 클수록 밝은 색을 가지므로 이 부분이 유사도가 가장 높다고 판단된 부분으로 인식한다. 유사도가 가장 높다고 판단된 부분의 좌표를 기반으로 원본 이미지에서 사각형을 그려보면 다음과 같은 결과가 추출된다.

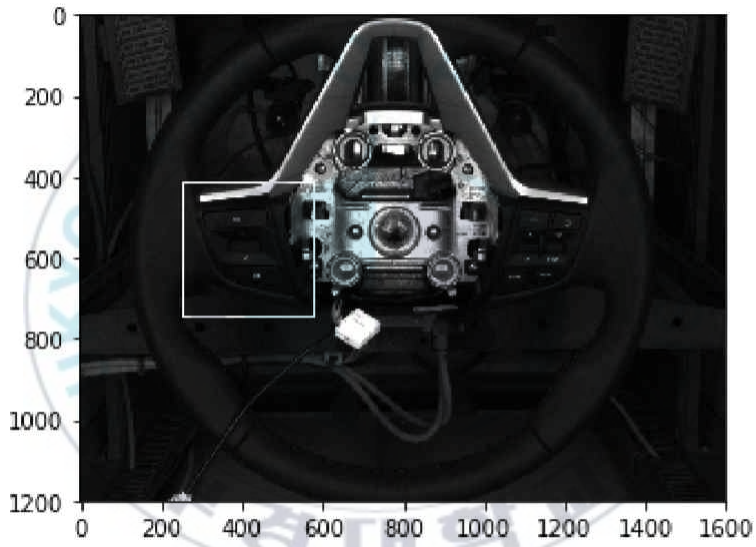


그림 17. 우측 이미지 템플릿 매칭 결과에 대한 사각형 표시

위와 같은 방법으로 좌측 스위치와 댐퍼에 대한 템플릿 매칭 결과는 다음과 같다.

그림 18. 좌측 스위치 템플릿 이미지

그림 18은 이전의 우측 스위치 템플릿 매칭을 위한 템플릿 이미지와 같이 좌측의 스위치를 전체 이미지에서 탐색하기 위한 이미지이다.

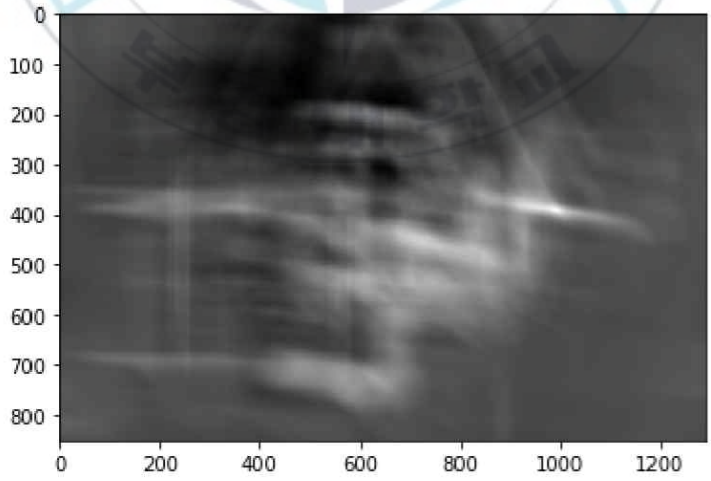


그림 19. 좌측 이미지 템플릿 매칭 결과

그림 19는 좌측 스위치에 대한 템플릿 매칭 결과이다. 우측 스위치와 동일하게 Correlation기반의 템플릿 매칭을 수행한다. 아래의 그림 20은 템플릿 매칭 결과를 원본 이미지에 표기한 것이다.

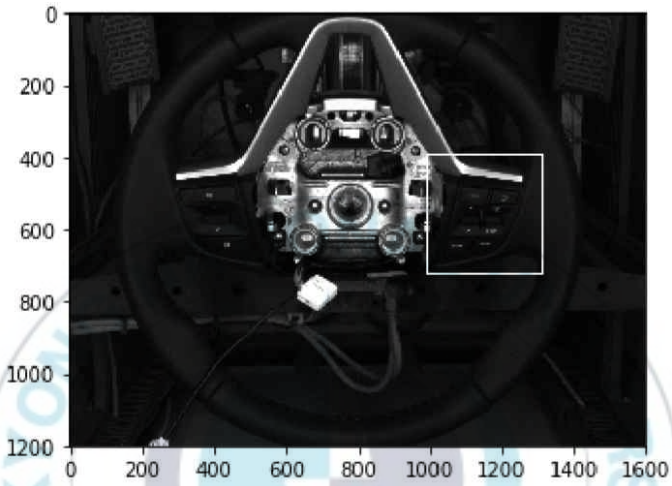


그림 20. 좌측 이미지 템플릿 매칭 결과에 대한 사각형 표시

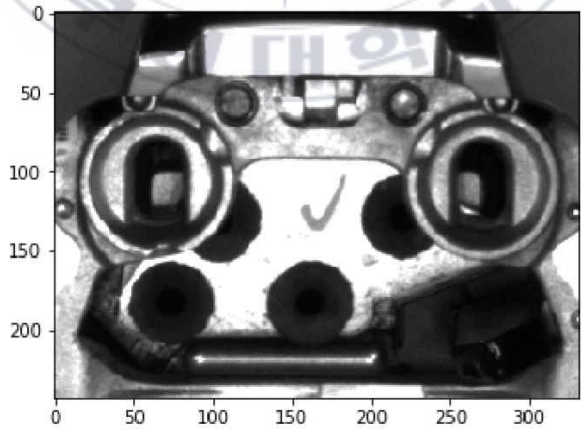


그림 21. 댐퍼 템플릿 이미지

그림 21은 댐퍼에 대한 템플릿 이미지다. 좌/우 스위치와 동일하게 기존 이미지에서 템플릿 이미지를 추출하고 다른 이미지들을 대상으로 템플릿 매칭을 수행한다. 그림 22는 댐퍼에 대한 템플릿 매칭 결과이다. 가운데 밝게 빛나는 부분을 찾을 수 있는데 이것이 픽셀단위 Correlation의 결과를 의미한다.

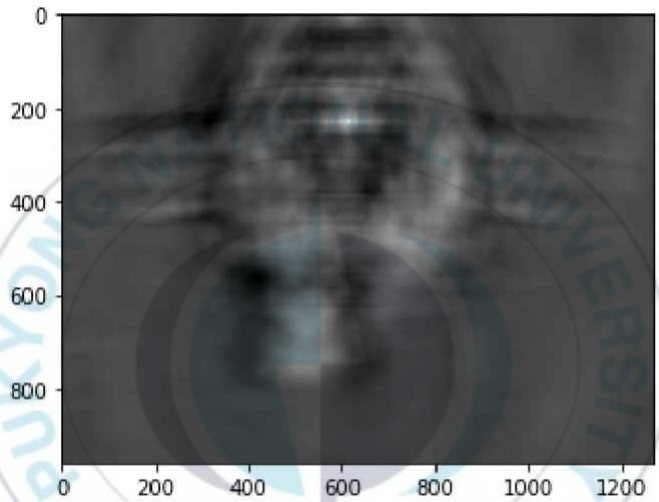


그림 22. 댐퍼 템플릿 매칭 결과

아래의 그림 23은 템플릿 매칭 결과를 원본 이미지에 표기한 결과이다. 모든 데이터를 대상으로 좌/우 스위치와 댐퍼의 템플릿 매칭을 수행하여 각 스티어링 휠을 구분하는 특성을 추출한다.

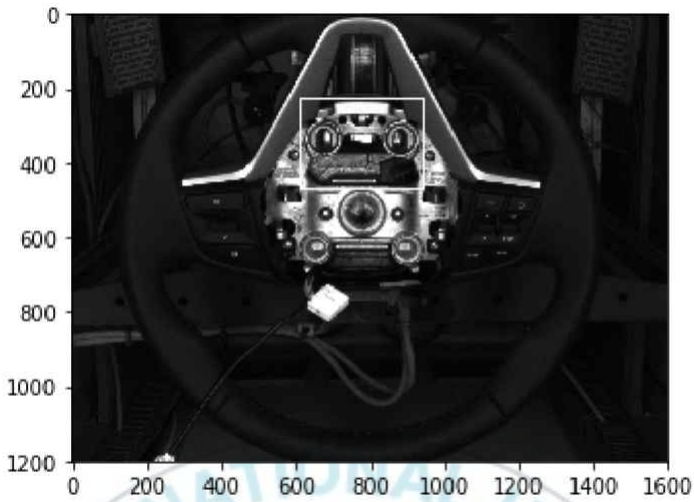


그림 23. 댐퍼 이미지 템플릿 매칭 결과에 대한 사각형 표시

4. 이진 분류와 다중 분류 모델

분류 범위 정의에서 n개의 자동차 스티어링 휠 분류를 5개의 분류로 축소 정의했다. 이는 분류할 스티어링 휠의 특성이 이미지에서 표현되는 특성 외에서도 나타나는 경우가 존재하고, 다른 분류의 휠이라도 같은 특성을 지니는 경우가 있기 때문이다. 본 연구에서는 5개의 클래스를 구분하는 스티어링 휠 분류모델을 개발하기 전에, 두 가지의 클래스를 선택해 이진 분류를 수행하는 모델을 개발한다. 이진 분류 모델을 먼저 개발하는 것은 두 가지의 이점이 있다.

첫 번째는 개발 하고자 하는 모델이 현재 구분하려고 하는 데이터의 표현(Representation)을 인식하는지 알 수 있다. 본 연구와 같이 딥러닝

모델의 경우 데이터를 기반으로 모델이 학습된다. 이때 모델은 데이터의 표현을 학습한다. 이미지 분류에 포함되는 스티어링 휠 분류 모델은 고차원의 이미지 데이터의 특징을 학습해야하기 때문에 이진분류 모델을 먼저 개발함으로써 분류에 대한 특성 엔지니어링과 함께 데이터의 표현을 확인 할 수 있다.

두 번째는 이진 분류 모델로 다중 분류 문제를 해결 할 수 있다. 다중 분류는 여러 개의 이진 분류 모델을 조합하여 수행 가능하다. 하지만 분류하려는 클래스가 너무 많은 경우에는 좋지 않은 접근법이다.

5. 전이 학습

전통적으로 머신러닝의 학습 알고리즘은 특정 문제에 국한해 학습된다. 즉 특정 도메인의 문제를 해결하기 위해 특정 데이터 사용해 학습한 결과 도메인, 데이터, 문제에 국한되어 모델이 학습된다.

전이학습은 1995년 신경 정보 처리 학회(Neural Infomation Processing System, NIPS, 현 NeurlPS)의 워크샵에서 발표한 "Learning to Learn : Knowledge Consolidation and Inductive Transfer(학습하기 위한 학습: 지식 통합 및 귀납적 전이)[4]"를 시작으로 발전되었다. 도메인, 데이터, 문제를 기반으로 학습된 모델이 한 가지 문제가 아닌 다른 설정의 일반화를 개선하기 위해 활용되는 것이다.

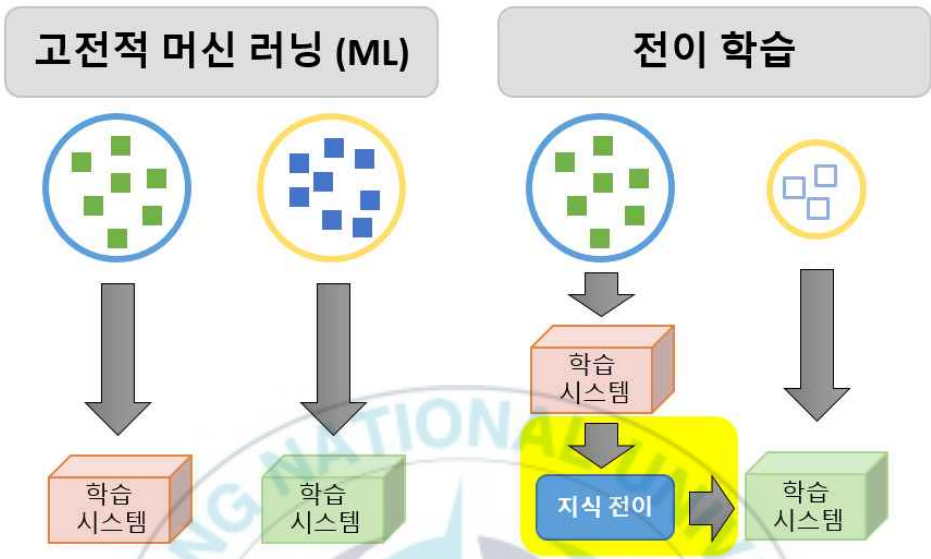


그림 24. 전이학습

전이학습은 다음과 같은 장점이 존재한다. 첫 번째, 향상된 기본 성능의 모델을 학습할 수 있다. 현재 타겟 모델뿐만 아니라 소스 모델의 지식에 학습을 시작하므로 기본 성능이 향상될 수 있다. 두 번째, 모델 개발 시간을 줄일 수 있다. 처음부터 랜덤화 된 가중치를 학습하는 것이 아닌 일반화된 모델의 가중치에서 시작한다. 즉 보다 적은 에폭[4]으로 최대 성능으로의 수렴이 가능하다. 세 번째, 향상된 최종 성능을 달성할 수 있다. 일반화 된 모델로부터 현재 타겟 모델의 데이터로 학습이 불가능한 특성들의 지식이 전이되기 때문에 최종 성능 또한 향상될 수 있다.

4) Epoch. 전체 Sample 데이터를 이용하여 한 바퀴를 돌아 학습하는 것을 1 epoch이라고 한다. 2 epochs는 데이터를 두 바퀴를 돌며 학습한 것을 뜻한다.

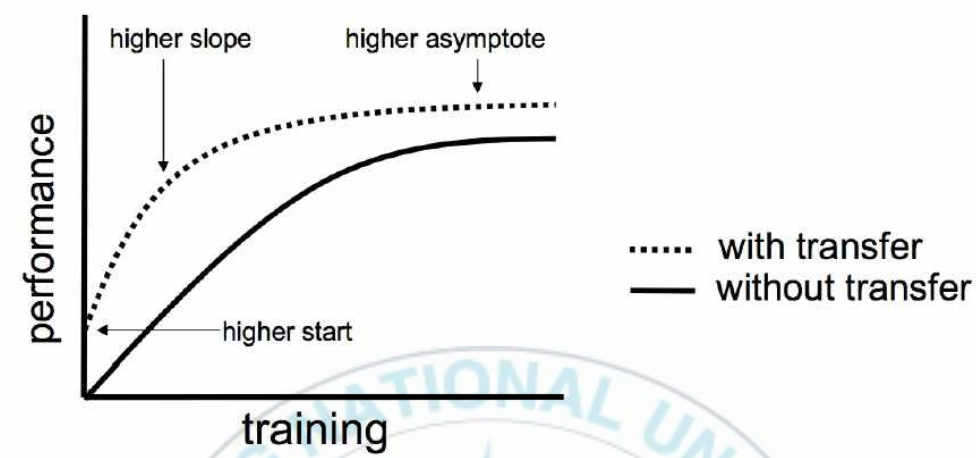


그림 25. 전이학습의 성능

본 연구에서는 스티어링 휠 분류 모델을 위해 전이학습을 다음과 같은 단계로 접근한다. 먼저 모델을 처음부터 학습해서 성능을 측정하고 이후 데이터 증폭에 의해 생성된 데이터와 함께 학습된 모델의 성능을 측정한다. 마지막으로 ImageNet 데이터로 사전 학습된 모델에 스티어링 휠 데이터를 학습한 후 해당 모델의 성능을 측정한다.

아래의 그림 26은 이번 장에서 정의한 스티어링 휠 분류 모델 개발에 대한 단계이다. 수집된 데이터를 특정 기준에 맞추어 분류할 클래스를 정의하고, 템플릿 매칭을 통하여 특정 요소를 탐색하고, 탐색한 부분을 CNN으로 학습하여 VGG, inception, ResNet, DenseNet으로 평가하고 데이터 증폭을 사용하여 또 평가한다. 그 다음, 전이학습을 사용하여 다시 평가한다.

그림 26. 스티어링 휠 분류 모델의 개발 단계

Ⅳ. 구현 및 성능 평가

이번 장에서는 이전 장에서 정의한 CNN기반 자동차 스티어링 휠 분류 모델을 구현하고 성능을 평가한다. 모든 구현 및 실험을 위해 Python 3.7 버전을 사용하고 대화형 환경에서는 주피터 노트북, 모델의 파이프라인 개발을 위해서는 Visual Studio Code 환경에서 사용한다. 데이터는 원격에 있는 리소스가 아닌 하드디스크에 저장되어 있는 상태로 접근한다. 이미지 데이터를 전처리 하기 위해 컴퓨터 비전 오픈소스 라이브러리인 OpenCV를 사용한다. OpenCV의 경우 C++버전과 Python 버전이 제공된다. 학습, 검증, 테스트 데이터 분리와 k폴드 검증을 위해 Scikit-Learn을 사용한다. CNN모델 구현을 위해 구글에서 개발한 오픈소스 딥러닝 프레임워크인 Tensorflow와 Keras를 사용한다. 다음은 본 연구에서 사용한 라이브러리 목록을 나타낸다.

표 5. 라이브러리 목록

라이브러리	용도
OpenCV	자동차 스티어링 휠 전처리(템플릿 매칭)
Numpy	Scikit-Learn, Tensorflow, Keras에서 사용 가능한 데이터 포맷으로 변환
Scikit-Learn	원-핫 인코딩, 학습/검증/테스트 데이터 분리
Tensorflow	딥러닝 모델 구현, 파이프라인 구현
Keras	사전학습 된 모델 사용

자동차 스티어링 휠 분류모델 구현에 앞서 실험과정을 단계별로 기술한다. 본 연구에서는 분류모델의 학습을 위해 처음부터 학습하는 기본 모델과 사전학습 된 모델을 사용한다. 사전학습 된 모델은 이전 장에서 살펴본 것과 같이 전이학습을 기반으로 하며 ImageNet 데이터로 사전학습 된 VGG의 11, 16 모델과 Inception, ResNet 각각으로 실험한다. ImageNet 데이터의 경우 2만 개의 클래스로 분류된 1,400만 장의 이미지 데이터이다. 개인이 이러한 볼륨의 데이터를 학습하는 것은 불가능하지만 Keras에는 ImageNet 데이터로 사전학습 된 모델, 본 연구에서 사용하는 VGG, Inception, ResNet 뿐만 아니라 MobileNet, DenseNet과 같은 여러 가지 CNN기반의 모델을 사용해 ImageNet 데이터 사전학습 된 모델을 제공한다.

1. 데이터 전처리

먼저 수집된 스티어링 휠 데이터에 대한 전처리를 진행한다. 수집된 스티어링 휠 데이터는 이미지 데이터이고 현재 비전 테스트기를 통해 수집되고 있다. 해당 비전 테스트기는 스티어링 휠이 상판에 고정된 상태일 때 이미지를 수집하여 대부분의 이미지가 동일한 각도로 고르게 수집된다. 수집된 데이터는 년도, 월, 일 별 디렉토리로 구분되어 있다.

본 연구에서는 스티어링 휠 분류를 위해 스티어링 휠의 종류를 구분할 수 있는 좌/우 스위치, 댐퍼 이미지를 사용한다. 이를 위해 년도, 월, 일로 구분된 데이터가 포함된 디렉토리를 순회하며 템플릿 매칭을 수행한다.

이후 각 픽셀에 대한 정규화를 진행한다. 정규화를 위해 Min-max 스케일링을 사용한다.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Scikit-Learn에서는 min-max 스케일링에 대한 함수를 지원한다.

2. 모델 학습

모델 학습을 위해 본 연구에서는 3가지의 접근법을 사용한다. 첫 번째는 가중치가 랜덤하게 초기화된 모델을 스티어링 휠 데이터로 처음부터 학습하고 평가하는 것이다. 두 번째는 ImageNet 데이터로 사전 학습된 모델에 마지막 분류기층만 다시 학습하는 것이다. 세 번 째 방법은 ImageNet 데이터로 사전 학습된 모델을 기반으로 분류지층뿐만 아니라 이미 학습된 가중치를 다시 학습하는 것이다. Keras에는 CNN기반의 각 모델들, 즉 VGG, InceptionV3, ResNet에 대한 ImageNet으로 학습된 모델을 제공한다. 다음은 각 모델에 대한 코드이다.

표 6. 분류 모델에 대한 Python 코드

```
import numpy as np
import tensorflow as tf

from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.layers import concatenate
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing import image



drop_out = 0.4
num_labels = 5
```

```
left_switch_input = Input(shape=(224, 224, 3))
right_switch_input = Input(shape=(224, 224, 3))
damper_input = Input(shape=(224, 224, 3))


left_switch_model = VGG16(input_tensor=left_switch_input,
weights='imagenet', include_top=False)
right_switch_model = VGG16(input_tensor=right_switch_input,
weights='imagenet', include_top=False)
damper_model = VGG16(input_tensor=damper_input, weights='imagenet',
include_top=False)


for layer in right_switch_model.layers:
    layer._name = layer._name + str("_left_switch")

for layer in right_switch_model.layers:
    layer._name = layer._name + str("_right_switch")

for layer in damper_model.layers:
    layer._name = layer._name + str("_damper")

out = left_switch_model.output
out = GlobalAveragePooling2D()(out)
out = Dense(512, activation='relu')(out)
left_switch_output = Dense(512, activation='relu')(out)

out = right_switch_model.output
out = GlobalAveragePooling2D()(out)
out = Dense(512, activation='relu')(out)
right_switch_output = Dense(512, activation='relu')(out)
```

```
out = damper_model.output
out = GlobalAveragePooling2D()(out)
out = Dense(512, activation='relu')(out)
damper_output = Dense(512, activation='relu')(out)

y = concatenate([left_switch_output, right_switch_output, damper_output])
y = Flatten()(y)
y = Dropout(drop_out)(y)
outputs = Dense(num_labels,  activation='softmax')(y)

model = Model([left_switch_input, right_switch_input, damper_input], outputs)

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

본 연구에서는 스티어링 휠 분류를 위해 스티어링 휠 이미지에서 추출한 좌측 스위치, 우측 스위치, 댐퍼 이미지를 사용한다. 각 이미지는 전처리단계에서 템플릿 매칭을 통해 추출한다. 분류모델은 먼저 각 이미지를 구분하는 CNN 모델이 구성되고 각 CNN모델의 출력을 연결하는 완전연결계층이 구성된다. 완전연결계층은 3~5개 층으로 이루어져 있고 마지막에는 분류를 수행하는 소프트맥스 계층이 구성된다.

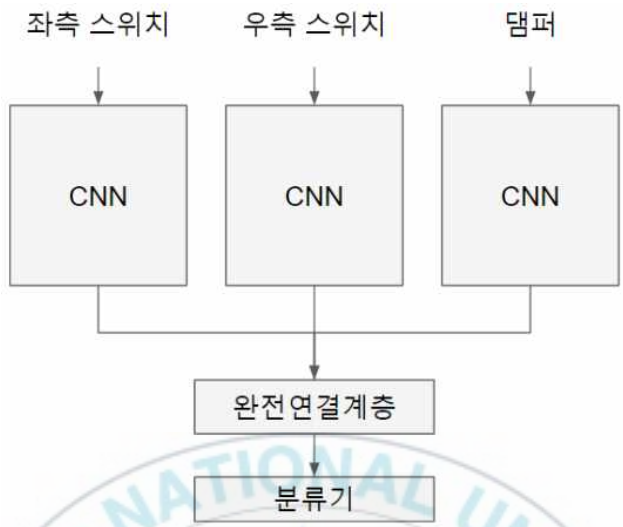


그림 27. 스티어링 휠 분류 모델 구조

이를 위해 Keras에서 제공하는 함수형 API를 사용한다. 함수형 API는 기존의 순차모델과 다르게 레이어의 입력과 출력을 모듈 단위로 구성 할 수 있도록 지원된다. 이를 이용하면 여러 모델을 합치거나 분리할 수 있다. 본 연구에서는 모델 학습과 테스트를 위해 8:2로 데이터를 분할하였다.

표 7. 본 연구의 학습과 테스트 데이터 모델 구조

Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	[(None, 224, 224, 3)	0	
input_52343_2_left_switch_right	[(None, 224, 224, 3)	0	
input_6_3_damper_damper (InputL	[(None, 224, 224, 3)	0	

block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792	input_4[0][0]
block1_conv1_left_switch_right_	(None, 224, 224, 64)	1792	input_52343_2_left_switch_right_s
block1_conv1_damper (Conv2D)	(None, 224, 224, 64)	1792	input_6_3_damper_damper[0][0]
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_conv2_left_switch_right_	(None, 224, 224, 64)	36928	block1_conv1_left_switch_right_sw
block1_conv2_damper (Conv2D)	(None, 224, 224, 64)	36928	block1_conv1_damper[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block1_pool_left_switch_right_s	(None, 112, 112, 64)	0	block1_conv2_left_switch_right_sw
block1_pool_damper (MaxPooling2	(None, 112, 112, 64)	0	block1_conv2_damper[0][0]
block2_conv1 (Conv2D)	(None, 112, 112, 128	73856	block1_pool[0][0]
block2_conv1_left_switch_right_	(None, 112, 112, 128	73856	block1_pool_left_switch_right_swi
block2_conv1_damper (Conv2D)	(None, 112, 112, 128	73856	block1_pool_damper[0][0]
block2_conv2 (Conv2D)	(None, 112, 112, 128	147584	block2_conv1[0][0]
block2_conv2_left_switch_right_	(None, 112, 112, 128	147584	block2_conv1_left_switch_right_sw
block2_conv2_damper (Conv2D)	(None, 112, 112, 128	147584	block2_conv1_damper[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block2_pool_left_switch_right_s	(None, 56, 56, 128)	0	block2_conv2_left_switch_right_sw
block2_pool_damper (MaxPooling2	(None, 56, 56, 128)	0	block2_conv2_damper[0][0]
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv1_left_switch_right_	(None, 56, 56, 256)	295168	block2_pool_left_switch_right_swi
block3_conv1_damper (Conv2D)	(None, 56, 56, 256)	295168	block2_pool_damper[0][0]
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv2_left_switch_right_	(None, 56, 56, 256)	590080	block3_conv1_left_switch_right_sw

block3_conv2_damper (Conv2D)	(None, 56, 56, 256)	590080	block3_conv1_damper[0][0]
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_conv3_left_switch_right_	(None, 56, 56, 256)	590080	block3_conv2_left_switch_right_sw
block3_conv3_damper (Conv2D)	(None, 56, 56, 256)	590080	block3_conv2_damper[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block3_pool_left_switch_right_s	(None, 28, 28, 256)	0	block3_conv3_left_switch_right_sw
block3_pool_damper (MaxPooling2	(None, 28, 28, 256)	0	block3_conv3_damper[0][0]
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv1_left_switch_right_	(None, 28, 28, 512)	1180160	block3_pool_left_switch_right_swi
block4_conv1_damper (Conv2D)	(None, 28, 28, 512)	1180160	block3_pool_damper[0][0]
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv2_left_switch_right_	(None, 28, 28, 512)	2359808	block4_conv1_left_switch_right_sw
block4_conv2_damper (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv1_damper[0][0]
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_conv3_left_switch_right_	(None, 28, 28, 512)	2359808	block4_conv2_left_switch_right_sw
block4_conv3_damper (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv2_damper[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block4_pool_left_switch_right_s	(None, 14, 14, 512)	0	block4_conv3_left_switch_right_sw
block4_pool_damper (MaxPooling2	(None, 14, 14, 512)	0	block4_conv3_damper[0][0]
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv1_left_switch_right_	(None, 14, 14, 512)	2359808	block4_pool_left_switch_right_swi
block5_conv1_damper (Conv2D)	(None, 14, 14, 512)	2359808	block4_pool_damper[0][0]
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]

```
block5_conv2_left_switch_right_ (None, 14, 14, 512)  2359808     block5_conv1_left_switch_right_sw
block5_conv2_damper (Conv2D)    (None, 14, 14, 512)  2359808     block5_conv1_damper[0][0]
block5_conv3 (Conv2D)           (None, 14, 14, 512)  2359808     block5_conv2[0][0]
block5_conv3_left_switch_right_ (None, 14, 14, 512)  2359808     block5_conv2_left_switch_right_sw
block5_conv3_damper (Conv2D)    (None, 14, 14, 512)  2359808     block5_conv2_damper[0][0]
block5_pool (MaxPooling2D)      (None, 7, 7, 512)    0           block5_conv3[0][0]
block5_pool_left_switch_right_s (None, 7, 7, 512)    0           block5_conv3_left_switch_right_sw
block5_pool_damper (MaxPooling2 (None, 7, 7, 512)    0           block5_conv3_damper[0][0]
global_average_pooling2d_9 (Glo (None, 512)          0           block5_pool[0][0]
global_average_pooling2d_10 (Gl (None, 512)          0           block5_pool_left_switch_right_swi
global_average_pooling2d_11 (Gl (None, 512)          0           block5_pool_damper[0][0]
dense_25 (Dense)                (None, 512)          262656      global_average_pooling2d_9[0][0]
dense_27 (Dense)                (None, 512)          262656      global_average_pooling2d_10[0][0]
dense_29 (Dense)                (None, 512)          262656      global_average_pooling2d_11[0][0]
dense_26 (Dense)                (None, 512)          262656      dense_25[0][0]
dense_28 (Dense)                (None, 512)          262656      dense_27[0][0]
dense_30 (Dense)                (None, 512)          262656      dense_29[0][0]
concatenate_9 (Concatenate)     (None, 1536)         0           dense_26[0][0]
                                                                 dense_28[0][0]
                                                                 dense_30[0][0]
flatten_9 (Flatten)             (None, 1536)         0           concatenate_9[0][0]
dropout_8 (Dropout)             (None, 1536)         0           flatten_9[0][0]
dense_32 (Dense)                (None, 5)            7685        dropout_8[0][0]
==================================================================================================
```

```
Total params: 45,727,685
Trainable params: 45,727,685
Non-trainable params: 0
_________________________________________________________________
```

위의 표 7은 본 연구에서의 학습에 사용된 모델의 구조이다. 조금 더 앞에서 제시했던 표 6의 코드를 살펴보면 본 연구에서는 VGG16 구조를 세 개 사용한 것을 알 수 있는데, 세 개의 병렬연결 구조를 텍스트로 표현한 것이 바로 표 7이다. 처음의 3개의 input은 그림 27에서의 각 input을 나타낸다. 그리고 input은 다른 block들과 연결되어 있는데, block 들을 살펴보면 Conv2D도가 있고 MaxPooling도 있다. 이것이 실제 VGG를 의미한다. 그리고 conncatnate가 3개의 dense 층을 이어서 flatten 층으로 넣고 있다. 표 7의 구조를 도식화 해둔 것이 그림 27이다.

3. 성능 평가

본 연구에서는 VGG, InceptionV3, ResNet50 총 세 개의 CNN구조를 사용한다. 또한 사전학습 되지 않은 모델과 가중치가 동결된 사전 학습된 모델(1) 그리고 가중치가 동결되지 않은 사전 학습 된 모델(2), 총 세 가지를 사용해 스티어링 휠 데이터를 분류하고 평가한다. 모든 모델의 비용 함수는 cross-entropy를 사용하고 최적화 함수로 Adam을 사용한다. 최적화 함수는 딥러닝에서 학습 속도를 빠르고 안정적이게 만들어

준다. 최근까지 개선된 최적화 함수들이 여럿 존재하지만, 본 논문에서 다루는 스티어링 휠의 이미지의 단순성에 가장 효율적인 함수는 Adam 이라고 판단하여 사용하였다.

표 8. 본 연구의 학습 모델 성능 평가 - 정확도

네트워크 모델 / 학습 유형	VGG16	InceptionV3	ResNet50
처음부터 학습한 모델	87.2 %	91.8 %	90.0 %
사전 학습 된 모델(1)	84.6 %	89.0 %	85.4 %
사전 학습 된 모델(2)	85.1 %	91.1 %	88.7 %

4. 웹 기반 데모 시스템 구현

본 연구에서 개발한 스티어링 휠 분류 모델을 처리하는 웹 기반 사용자 인터페이스를 제시한다. 웹 기반 사용자 인터페이스는 실제 적용 환경에 배치된 기존의 비전 테스트기 패널이나 작업자들이 사용 중인 디스플레이에 대한 호환성을 고려한 사항이다. 작업자가 테스트 결과를 볼 수 있도록 반드시 결과가 화면에 출력되어야 하고, 제품의 실측된 이미지 역시 화면에 표시가 되어야 이중으로 확인 할 수 있다.

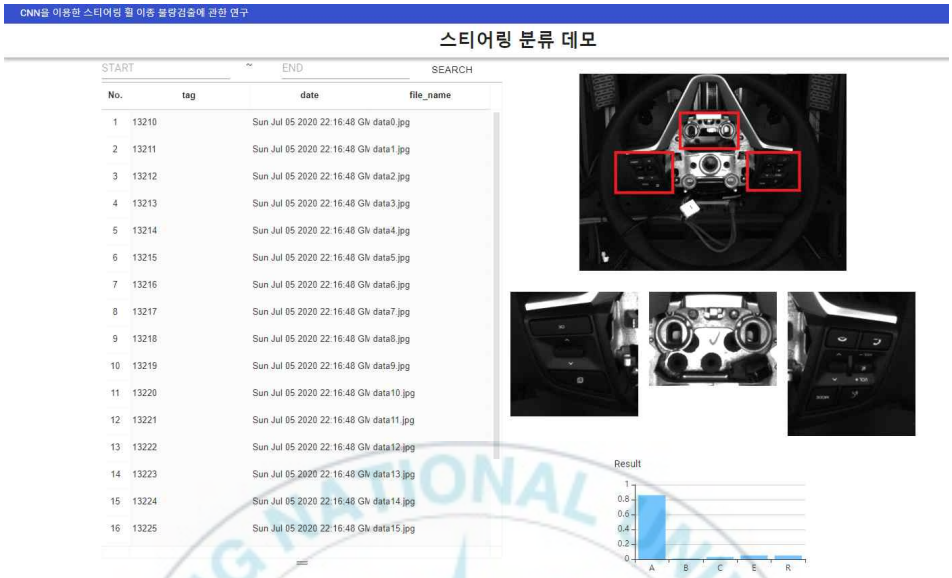


그림 28. 웹 기반 데모 시스템의 UI

그림 28은 위에서 언급한 조건에 맞추어 실제로 구현해 본 화면이다. 좌측의 목록은 비전 테스트기가 수집하는 스티어링 휠의 데이터를 의미한다. 우측 상단은 생산 중인 스티어링 휠에서 좌/우 스위치와 댐퍼를 템플릿 매칭한 결과이고 우측 하단은 수집된 데이터의 스티어링 휠이 본 연구에서 제시한 다섯 가지 클래스(A, B, C, E, R) 중 어느 클래스에 속하는 것인지에 대한 분류 결과를 의미한다.

실제 조립 공장에서 쓰이고 있는 비전 테스트기에 적용을 아직 해보지는 못했지만, 적용을 하게 된다는 것을 가정해 보았을 때의 프로세스는 아래와 같이 진행된다.

비전 테스트기에 제품을 안착 → 이미지 촬영 → OpenCV를 활용한 템플릿 매칭으로 특성을 나타내는 부위 추출 → CNN을 통하여 학습된 모델의 분석 → 화면에 결과 출력 → 사용자의 최종 승인

그리고 아래의 그림 29에 표시해둔 것처럼 스티어링 휠의 바디 종류에 대한 학습 모델과 베젤의 컬러를 판단하는 학습 모델도 추가하면 본 연구에서 제시한 다섯 가지의 클래스 대신 160여 종이 넘는 세부 클래스를 판단하여 결과를 제시 할 수 있을 것이다. 추가로 볼트의 체결 여부까지 체크할 수 있는 학습 모델까지 추가하면 이종 불량 외에도 누락 불량까지도 같이 체크하여 사용자에게 결과를 출력하여 제시할 수 있게 된다면 불량으로 인한 손실 비용을 더욱 더 줄일 수 있을 것이다.

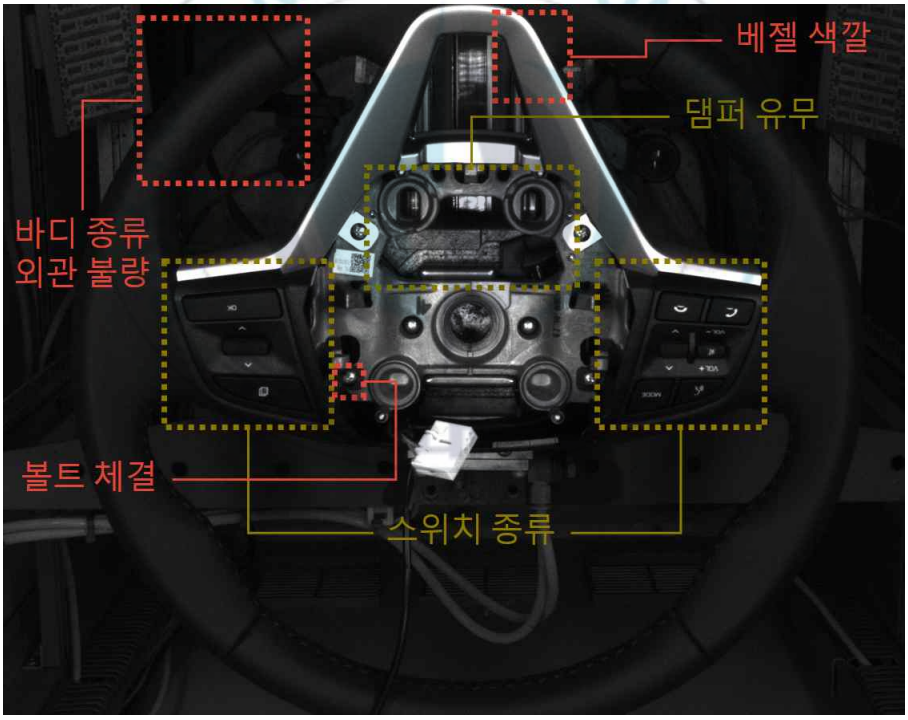


그림 29. 스티어링 휠 향후 연구 방향 요소

Ⅴ. 결론

1. 결론 및 고찰

본 연구에서는 CNN기반의 자동차 스티어링 휠 분류 모델을 정의하고 구현 및 실험 결과를 제시했다. 모델을 정의하기 위해 3개의 CNN 구조를 사용했다. 여기에는 VGG, Inception, ResNet이 해당하며 딥러닝 오픈소스인 Tensorflow를 활용해 구현했다. 학습을 위해 처음부터 학습하는 모델과 함께 전이학습 기반의 사전학습 된 모델을 사용했다. 여기서 사전 학습된 모델이란 ImageNet 데이터를 기반으로 학습된 모델을 의미하며 Tensorflow에서 제공되는 사전 학습 모델을 의미한다. 모델의 인풋으로 스티어링 휠 전체 이미지에서 스티어링 휠을 분류하는 3가지 부분인 좌/우측 스위치와 댐퍼 부분을 사용했다. 이를 위해 컴퓨터 비전 오픈 소스인 OpenCV를 사용해 템플릿 매칭을 수행했다. 이미지에 표현되는 스티어링 휠 특징의 제한으로 총 160개의 분류를 5개의 분류로 레이블링한다. 결과적으로 본 연구에서 제시하는 스티어링 휠 분류 모델은 스티어링 휠에 표현되는 3개의 부분을 각각의 이미지로 입력으로 받아 5개의 클래스로 분류한다. 학습에서 비용함수로 cross-entropy를 사용하고 최적화함수로 Adam을 사용했다. VGG모델의 경우 84%~87%의 정확도를 보였으며 Inception은 89%~91%, ResNet의 경우 85%~90%의 정확도를 보였다.

알파고를 통해 대중에게 널리 알려진 인공지능을 이용하여 실무에서 쓰이고 있는 기존 설비의 단점을 보완 할 수 있는 방향에 대해 제시했다는 점에서 본 논문이 시사하는 바는 상당하다고 생각한다. 이미 딥러닝에 관한 많은 연구가 이루어져있지만, 실제 제조업 분야에서 적용되고 있는 케이스는 아주 드물다고 생각한다. 그리 크지 않은 비용으로도 충분히 도입 가능한데다가, 본 논문의 내용을 바탕으로 조금 더 발전시켜서 제조업의 품질관리 업무에 실제로 적용시킬 수 있다면 투자비용 대비 효율성의 측면에서 아주 합당 할 것이라고 생각한다.

2. 추후 연구

본 연구에서 100% 혹은 그에 근접한 수치의 정확도를 기대했으나, TL이라는 한 개의 차종만의 데이터로 한정지어 학습을 진행한 부분과 160여 종류의 사양을 다섯 개의 클래스로 묶는 과정에서의 조건 등으로 인하여 기대를 완벽하게 충족시키지는 못하였다. 다양한 차종의 충분한 데이터를 더 수집하고 클래스 분류의 기준을 조금 더 분명하게 구분하여 진행한다면 100%의 정확도를 가질 수 있을 것이라 생각한다. 또는 향후 연구에서는 별도의 클래스로 묶지 않고 각각의 사양으로 분류하여 구분해낼 수 있도록 학습을 시키는 방향으로 발전되어야 할 것이다.

실제 적용될 시스템에 대한 설명에서 간략히 언급하였지만, 위의 그림 29에서와 같이 빨간색으로 표시 된 부분은 향후에 연구 할 가치가 충분한 요소라고 생각한다. 본 논문에서는 스위치의 사양 이종과 댐퍼의 장

착 유무에 관해서만 집중적으로 다루었지만, 조립 과정에서 간혹 볼트 체결을 누락하고 가조립만 된 채 공정을 통과하는 경우라던가, 베젤의 컬러를 헷갈려서 이종 조립을 하는 경우의 불량 검출에 대한 추가 연구도 진행해 볼 가치도 충분히 존재한다. 실제로 누락과 컬러이종의 불량도 많이 발생하고 있으며, 본 논문에서 중점적으로 다룬 사양 이종과 같이 OpenCV의 이미지 템플릿 매칭 방식의 기존 비전 테스트기로는 불량의 검출이 어려워서 사람이 직접 최종 검사를 진행하고 있다.

그리고 PU발포 그립과 가죽 그립, 열선 그립, 반천공 그립 등 다양한 바디 사양의 구분과 함께 가죽 제품의 외관에 대한 품질 검사까지 함께 가능 할 것으로 예상한다. 가죽 제품의 특성 상, 천연 결함(힘줄, 모기물린 자국, 눌린 자국 등)과 납품 과정에서 생긴 결함 등을 세밀하게 판별하여야 하는데, 이는 현재 인력으로 100% 검사하고 있다. 추가 연구를 진행하여 실무에 적용할 수 있다면 스티어링 휠 업체 뿐 아니라 암레스트, 콘솔박스 등 가죽 소재를 다루는 자동차 내장 부품업체를 비롯한 모든 제조업 전반에 큰 도움이 될 것이라 확신한다. 가죽 소재 뿐 아니라, 플라스틱 사출물이나 다이캐스팅 된 제품의 외관 불량 품질검사 업무에도 역시 도움이 될 것이다.

초기의 학습을 위한 데이터를 수집하는 것이 가장 큰 관건이겠지만 전이학습과 추가 연구를 통해 보완한다면, 흔히들 감성품질[5]이라고 부르는 품질한도가 적용되고 있는 곳이라면, 사람이 주축이 되어 언제나 휴먼에러의 발생 가능성을 내포하고 있는 위험에서 벗어나 양품과 불량의 판정에 완벽함을 추구할 수 있게 되기 때문에 생산품의 품질 검사 방식과

5) 감성품질이란, 기능상으로는 아무런 문제가 없으나 외관상의 불량을 의미하기도 하고, 외관 불량에 대한 판정 기준을 명확하게 정의하기 힘든 소재 또는 요소를 가지고 있는 부품에 대한 품질 기준을 말한다.

개념 자체를 바꾸어 품질 관리 업무의 헤게모니를 가져 올 수도 있을 것이라고 생각한다.

< 참 고 문 헌 >

[1] 김영규, 박태형, “딥러닝을 이용한 표면 실장 부품의 PCB 조립 결함 분류", 제어로봇시스템학회, pp.209~410, 2017.

[2] 김대현, 부승빈, 홍현철, 여원구, 이남용, “딥러닝 알고리즘을 이용한 머신 비전 기반 불량 검출 연구”, 비파괴검사학회지 제40권 제1호, pp. 47~52, 2020.

[3] A methodology for part classification with supervised machine learning, https://www.cambridge.org/core/journals/ai-edam/article/methodology-for-part-classification-with-supervised-machine-learning/69D95B66344317AE778C1058993BC2B9/core-reader

[4] Sinno Jialin Pan, Qiang Yang, “A Survey on Transfer Learning”, IEEE Transactions On Knowledge and Data Engineering, Volume 22, pp. 1345~1359, 2019.

[5] Y LeCun, Y Bengio, G Hinton, “Deep learning”, Nature 521 (4553), pp. 436~444, 2015.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE, pp. 1~46, 1998, 12.

[7] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks” Advances in Neural Information Processing System25, Volume1, pp. 1097~1105, 2012.

[8] Karen Simonyan, Andrew Zisserman (2015) “Very Deep Convoluti

onal Networks for Large-Scale Image Recognition", International Conference on Learning Representations, arXiv:1409.1556, 2015. 4.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, A ndrew Rabinovich, "Going deeper with convolutions", Computer V ision and Pattern Recognition", arXiv:1409.4842, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Res idual Learning for Image Recognition", Computer Vision and Pat tern Recognition, arXiv:1512.03385, 2016.

[11] Supervised Learning, https://en.wikipedia.org/wiki/Supervised_lear ning

[12] Unsupervised Learning https://en.wikipedia.org/wiki/Unsupervised_l earning

[13] Online Machine Learning, https://en.wikipedia.org/wiki/Online_mac hine_learning

[14] Template Matching using OpenCV, https://docs.opencv.org/2.4/doc /tutorials/imgproc/histograms/template_matching/template_matchin g.html

[15] MLOps, https://en.wikipedia.org/wiki/MLOps

[16] Studer, S., Bui, T.B., Drescher, C., Hanuschkin, A., Winkler, L., P eters, S. and Mueller, K.R. (2020) "Towards CRISP-ML (Q): A Machine Learning Process Model with Quality Assurance Metho dology". arXiv:2003.05155, 2020

감사의 글

이 석사 학위 논문을 작성하는데 주변의 많은 지원과 독려와 가르침이 있었습니다. 바쁘신 가운데에도 항상 배려와 격려로 지도하여 주신 김창수 교수님께 먼저 진심으로 감사를 드립니다. 다른 학교에서 도시공학을 전공으로 학사를 졸업하고 서른한 살의 나름 늦은 나이로 컴퓨터와 프로그래밍에 대해 석사 공부를 해보겠다고 뛰어든 제자를 이렇게 이끌어주심에 감사하고 죄송할 따름입니다. 그리고 부족한 저를 너무 좋게 봐주시고 제 평생의 동반자까지 만나게 해주신 황현숙 박사님, 인자하신 미소로 조언을 아끼지 않으셨던 박만곤 교수님과 김명희 교수님께도 진심으로 고맙습니다.

머신러닝이라는 낯선 분야에 대해 알게 해주고 큰 도움을 준 서영원과 언제나 날 응원해주는 고윤성을 비롯한 우리 UPSIL 연구실 동생들, 힘든 일이 있거나 좋은 일이 있으면 항상 모여서 서로 힘이 되어주는 너희들 덕분에 이 논문을 쓸 수 있었다고 생각한다. 고맙다. 중학교 시절부터 내겐 늘 힘이 되어주는 강윤성, 맛있는 음식을 먹을 때면 먼저 떠오르는 안대희, 김성호, 김준엽을 비롯해 다들 그 누구보다도 멋지게 잘 살고 있는 새부산학원 친구들도 모두 고마워. 그리고 연구를 위해 수시로 자료를 제공해준 우리 보현MTS의 박석찬 대리와 언제나 회사를 위해 힘써주고 계신 박범진 부장님 이하 모든 직원 여러분들께도 감사의 인사를 전합니다. 항상 제 곁에서 저를 돌봐주고 도와주시는 고모부께도 특히 고맙습니다. 일과 공부에 지쳐 기분전환을 하고 싶을 때면 나와 같이 잘 놀아주는 궁디팡팡 녀석들에게도 고맙다는 말을 하고 싶습니다. 그리고 저 자신을 돌아볼 수 있게 도와주신 김병조 베드로 신부님과 박민아 로

사 선생님께도 너무 고맙고 감사합니다.

마지막으로, 항상 옳은 길로 이끌어 주시고 묵묵히 믿고 지켜봐주시는 사랑하는 부모님과 장인어른과 장모님께도 사랑과 고마움과 죄송함을 전합니다. 부모님들이 계시지 않았다면 지금 제가 없었을 것입니다. 평소에 표현은 잘 하지 못하지만 항상 잊지 않고 살고 있습니다. 고맙습니다. 건강하게 오래오래 저희 곁에 계셔주셨으면 좋겠습니다. 그리고 하나 밖에 없지만 딱히 아끼진 않는 내 여동생 산솔이에게도 아주 작은 사랑과 감사를 보냅니다. 그리고 가장 중요한, 이제 두어 달 뒤에 태어날 우리 건강한 복덩이, 강복이와 함께 늘 나를 응원해주고 지지해주는 사랑하는 아내 김서연 교수에게 무엇보다도 큰 사랑과 감사를 전합니다.