Occupant Detection by Infrared Images Based on Convolutional Neural Network

2017.12.26

김상혁(20170145)
Display System Design Lab.
Sogang University

Outline

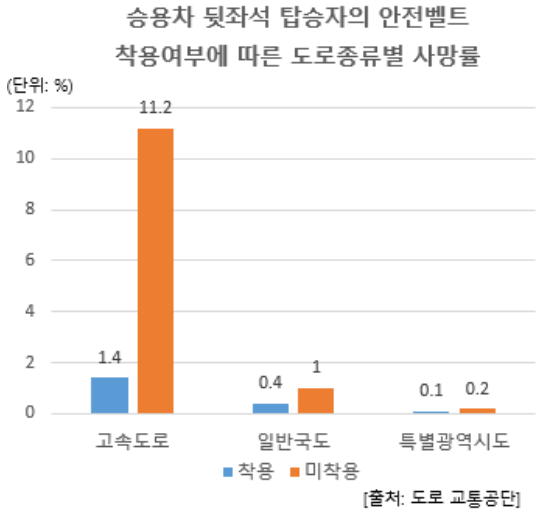
- Introduction
- Issues
- Related works
- Proposal direction
- Proposed system
- Implementation
- Experimental results
- Conclusion
- Future works
- Reference

IR 카메라를 이용한 영상 데이터 베이스 촬영

Introduction

- 안전벨트 착용 의무화
  - 정보통신산업 진흥원에 따르면 자동차 관련 국제 기준을 정립하는 WP29에서 자율주행 Level 2~3 자동차 내·외부 안전 표준 기준 정립 중
  - 도로 교통공단의 정보에 의하면 안전벨트 미착용시 사망률이 높음
    - 삼성교통안전문화연구소에 따르면 안전벨트 미착용 시 교통사고 치사율(2.4%)은 착용시(0.2%)보다 12배 높음
  - 2017년 6월 이후 전 좌석 안전벨트 착용 의무화

수준	정의	개요
Level 0	수동	• 운전자가 안전에 민감한 기능을 제어 및 교통 모니터링에 대한 안전 조작에 책임짐
Level 1	기능 제한 자동	• 운전자가 정상적인 주행 혹은 충돌 임박 상황에서 일부 기능을 제외한 자동차 제어권 소유
Level 2	조합 가능 자동	• 두 개 이상의 제어 기능이 조화롭게 동작 • 운전자가 모니터링 및 안전에 책임짐
Level 3	제한된 자율주행	• 특정 교통환경에서 자동차가 모든 안전 기능 통제 • 운전자 제어가 필요한 경우 경보신호 제공 • 운전자는 간헐적으로 제어
Level 4	완전 자율주행	• 자동차가 모든 안전 기능을 통제 • 운전자는 목적지 혹은 운행을 입력 • 자율주행 시스템이 안전운행에 책임짐



Issues

- 기존 시스템의 문제점
  - 탑승자 관련 차량 내 모니터링 Database의 부재
    - 차량 내부에 대한 영상 Database 생성 시 표준 절차 부재
  - 운전자 영상 획득 시 외부환경 민감
    - 기존 RGB 카메라로 인한 단점
  - 중량 센서를 통한 탑승자 유무 판단
    - 방치 승객 검출 및 물체에 대한 오검출 발생

여름날 방치된 아이(방치 승객)의 사망 사건이 빈번히 발생

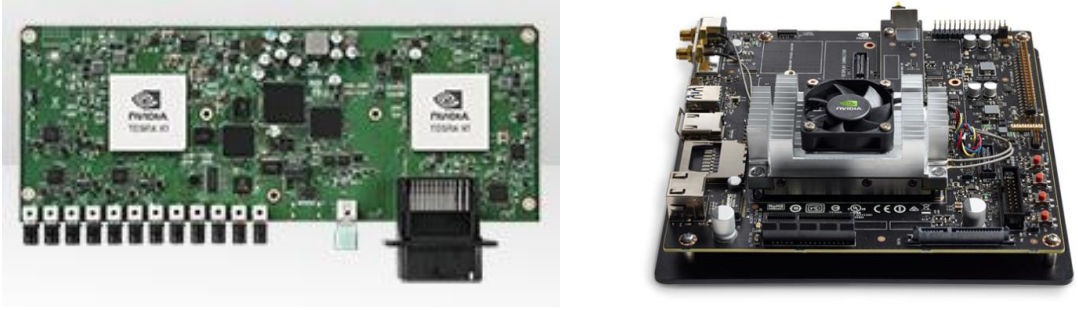
Related works

- 딥러닝 알고리즘의 발전
  - Imagenet competition을 중심으로 여러 Architecture 발달
    - VGGNet[1], ShuffleNet[2]등
  - 다양한 Application에서 이용
    - Application에 알맞은 형태의 DB 필요
  - 데이터 학습 시 기존 Machine learning 알고리즘 대비 많은 양의 데이터 필요
    - Eg. Imagenet: 14,197,122 영상(21,841 class)
  - 제안된 데이터 양에 대한 Augmentation 알고리즘 필요
    - GAN(Generative Adversarial Netoworks)[3]

[1] K. Simonyan, et al., "Very Deep Convolutional Networks for Large-Scale Image Recognition," Computer Vision and Pattern Recognition, 2015.
[2] X. Zhang, et al., "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," Computer Vision and Pattern Recognition, 2017.
[3] I. J. Goodfellow, "Generative Adversarial Networks," Machine Learning, 2014.

Proposal direction

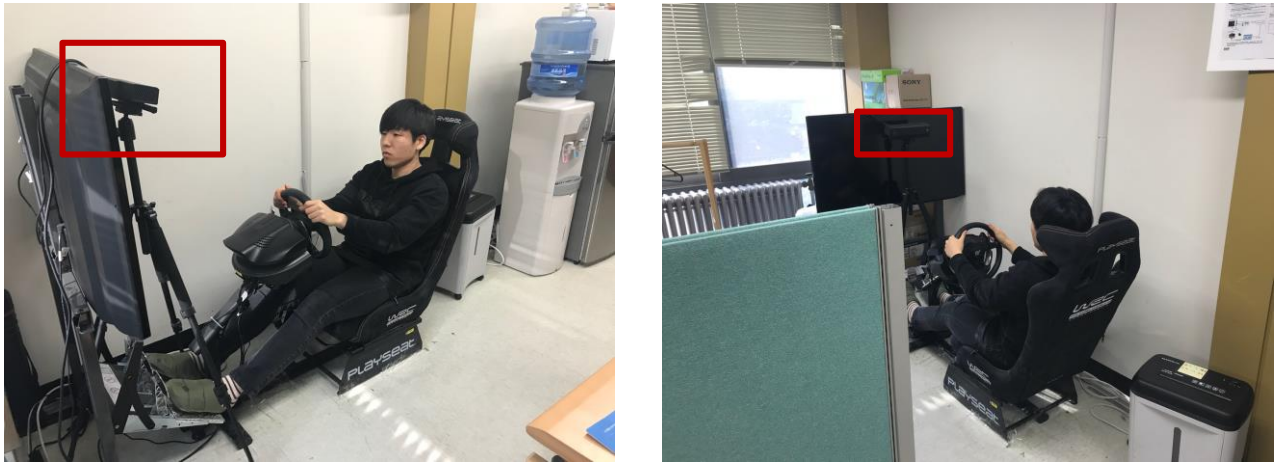
- IR 영상을 이용한 Deep neural network 기반 탑승객 검출 알고리즘 설계
  - 주·야간에 강건한 동작을 위한 IR 영상 기반의 검출 및 분류기 개발
- 실차 환경 조성을 통해 알고리즘에 적합한 Data set 구성
  - 차량 환경 조성 및 촬영 매뉴얼 제작을 통한 정형화를 통한 IR Data set 제작
- 인공 신경망 학습을 위한 충분한 수의 데이터 확보
  - 습득된 IR 영상에 대하여 Computer vision algorithm 적용을 통한 Data augmentation 실시
- Embedded 보드에 구동 가능한 Real-time process에 적합한 코딩
  - 기존에 존재하는Architecture를 기본 제공 library만을 이용한 C 기반의 코딩 실시



GPU가 내장된 Embedded board인 Driver PX(좌, 약 500만원)와 NVIDIA TX(우, 약 80만원)

Proposed system; DB construction

- 자체 DB 제작 환경
  - Driving simulator에서 촬영
    - 실제 운전 상황 도출
  - 현대모비스 요청 유사 위치에 카메라 설치
    - 차량내 카메라 설치 환경 고려



Driving simulator를 이용한 탑승객 DB 구축 환경

Proposed system; DB construction

- 시나리오 적용한 다양한 DB 제작
  - 12명의 연구원에 대하여 실시
    - 운전석에 대한 DB 구축
      - Computer vision 알고리즘을 이용하여 DB 확대
  - 자체 제작 시나리오 적용 및 예외 사항 적용
    - 기존 시나리오 기반한 데이터베이스 구축
    - 피험자에게 일정 수준의 자유로운 동작을 허용하여 여러 예외 상황 도출
      - 책 읽기, 겉옷 갈아입기


(a)


(b)


(c)


(d)

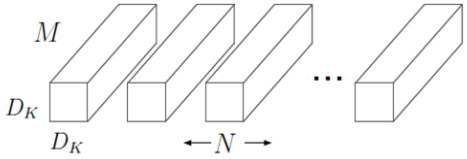
데이터 베이스 구성 예시: (a) 운전, (b)휴대폰 사용, (c)몸의 일부분 노출, (d) 책 읽기 (예외 상황)

Proposed system; MobileNet-28

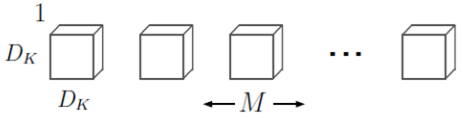
- 다양한 딥러닝 Architecture 구현 및 확인
  - Embedded 환경에 최적화된 Network 선택
    - Memory inference issue
    - Real-time system issue
  - Customized DB에 적합한 Network 축소
    - 적외선 영상을 통한 탑승객 유무 판단
      - 기존 ImageNet(RGB)와의 차이점 인지
  - 실제 구현 시 문제점 파악
    - Reference에 들어나지 않은 예외적인 문제 상황에 대한 대비
      - 문제 발견 시 Network 수정 또는 변경 실시

Proposed system; MobileNet-28

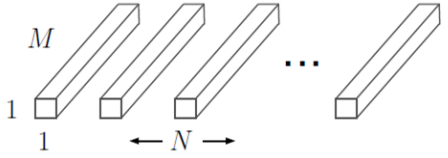
- MobileNet [4]
  - Depth/Point-wise convolution filter 사용
    - Depth/Point-wise convolution filter를 연속적으로 사용
      - Parameter 수 감소를 통한 Computation cost 감소
  - Average pooling layer를 사용
    - 전체 Convolutional layer 이후 실시
      - Fully-connected layer를 1개로 감소하여 Parameter 수 감소
  - Batch normalization layer 사용
    - 평균 0, 표준편차 1인 Feature 분포로 정규화
      - 각 Layer 마다 입력 feature의 분포가 달라지는 현상(Internal covariance shift)으로 Gradient vanishing/exploding 발생을 막음
    - Dropout layer 및 Weight regularization term 제외
      - 학습 속도 향상



(a) Standard convolution filter



(b) Depth-wise convolution filter



(c) Point-wise convolution filter

[4] A. G. Howard, et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv, 2017.

Proposed system; MobileNet-28

- MobileNet body architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Proposed system; MobileNet-28

- MobileNet-28 주요 Parameter 구성
  - Gradient decent type: Adam
    - 보편적인 방법 채택
  - Learning rate policy: Polynomial decay
    - 학습률 변화를 통한 최적 값 탐색

$$base\_lr_n = base\_lr_{n-1} \times (1 - \frac{iter}{\max\_iter})^{power}$$

$base\_lr_n$: n번째 학습률 $base\_lr_{n-1}$: $(n-1)$번째 학습률 $power$: power

$iter$: iteration 횟수 $\max\_iter$: 총 iteration 횟수

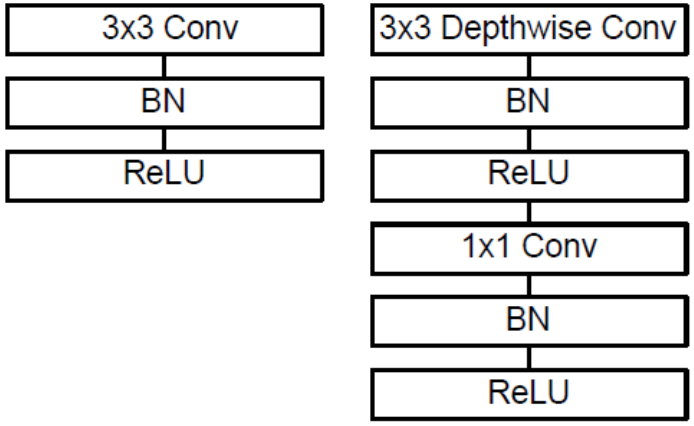
  - Weight initialization: MSRA (Microsoft Research Asia) [5]
    - Symmetry를 제거하는 방향으로 초기화
    - Xavier (일반적) 방법에 비해 성능이 뛰어난 방법 중 보편적인 방법 채택

Init method	maxout	ReLU	VLReLU	tanh	Sigmoid
LSUV	**93.94**	**92.11**	92.97	89.28	n/c
OrthoNorm	93.78	91.74	92.40	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	**93.09**	–	n/c
Xavier	91.75	90.63	92.27	**89.82**	n/c
MSRA	n/c†	90.91	92.43	89.54	n/c

[5] K. He, et al., "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," ICCV, 2015.

Proposed system; MobileNet-28

- MobileNet-28; Caffe 기반 학습 코드
  - 단순한 형태의 Layer 쌓기를 통한 학습기 구현 가능
  - 자율성이 상대적으로 제한
    - Library 이외의 주체적 Layer 구현에 제한



(a) (left) standard Conv layer. (right) Depthwise separable Conv layer.

```
name: "MobileNet"
layer{
   name: "dsd"
   type: "Data"
   top: "data"
   top: "label"
   include{
      phase: TRAIN
   }
   transform_param{
      scale: 0.00390625
   }
   data_param{
      source: "examples/hyundai_mobis/hm_train_lmdb"
      batch_size: 64
      backend: LMDB
   }
}
```

```
layer {
  name: "conv2_2/dw"
  type: "Convolution"
  bottom: "conv2_1/sep"
  top: "conv2_2/dw"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  convolution_param {
    num_output: 32
    bias_term: false
    pad: 1
    kernel_size: 3
    group: 32
    #engine: CAFFE
    stride: 2
    weight_filler {
      type: "msra"
    }
  }
}
```

(b) (left) MobileNet 초기 세팅. (right) Depthwise separable Conv layer.

Proposed system; MobileNet-28

- MobileNet; Keras 기반 학습 코드
  - 단순한 형태의 Layer 쌓기를 통한 학습기 구현 가능
  - 자율성이 상대적으로 보장
    - Tensorflow를 backend로 구성되어 Customized layer 구현 가능

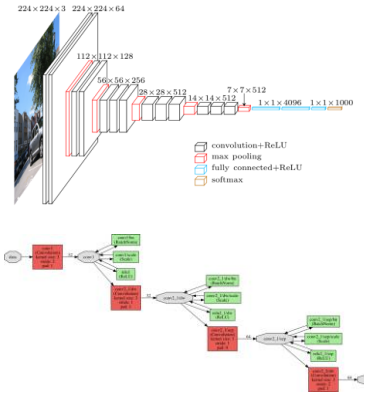
```
x = Convolution2D(int(32 * alpha), (3, 3), strides=(2, 2), padding='same', use_bias=False)(img_input)
x = BatchNormalization()(x)
x = Activation('relu')(x)

x = SeparableConv2D(int(32 * alpha), (3, 3), strides=(1, 1), padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Convolution2D(int(64 * alpha), (1, 1), strides=(1, 1), padding='same', use_bias=False)(x)

x = BatchNormalization()(x)
x = Activation('relu')(x)
```

MobileNet Conv layer.

Proposed system; occupant detection

- 딥러닝 분류기 기반 탑승객 검출 알고리즘
  - 학습된 Weight를 기반으로 제작
    - Offline learning을 기반 실시간 시스템 구현
  - Computational complexity 최적화 구현
    - C기반 시스템 구현

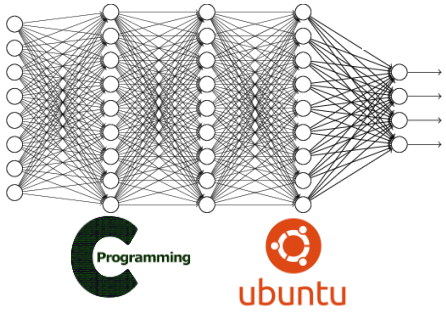
```
//#pragma once
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include<conio.h> // For exiting program by using ESC button
#include <windows.h> // Load images (only for Windows OS)
#include <time.h>  // Fot time-stamp

// Load images (only for Windows OS)
BITMAPFILEHEADER    Header_File;
BITMAPINFOHEADER    Header_Info;
```



Occupant detection based CNN

$$\begin{bmatrix} W_{00} & \cdots & W_{n0} \\ \vdots & \ddots & \vdots \\ W_{0m} & \cdots & W_{nm} \end{bmatrix}$$

Load weights of active functions



Feed forward network

Real-time process in vehicle

Proposed system; occupant detection

- Memory allocation 및 release
  - 영상 데이터 입력 전 memory allocation
    - 전역 변수에 선언
      - 동적 할당
  - 시스템 종료 전 memory release

```
// Allocate depth-wise convolutional filters
fConv_1_dw = fTri_Arr(iChannel_01*iAlpha_Width, 3, 3); // 3 x 3 x 32 dw
fConv_2_dw = fTri_Arr(iChannel_02*iAlpha_Width, 3, 3); // 3 x 3 x 64 dw
fConv_3_dw = fTri_Arr(iChannel_03*iAlpha_Width, 3, 3); // 3 x 3 x 128 dw
fConv_4_dw = fTri_Arr(iChannel_03*iAlpha_Width, 3, 3); // 3 x 3 x 128 dw
fConv_5_dw = fTri_Arr(iChannel_04*iAlpha_Width, 3, 3); // 3 x 3 x 256 dw
fConv_6_dw = fTri_Arr(iChannel_04*iAlpha_Width, 3, 3); // 3 x 3 x 256 dw
// Iteration
fConv_7_dw = fTri_Arr(iChannel_05*iAlpha_Width, 3, 3); // 3 x 3 x 512 dw
fConv_8_dw = fTri_Arr(iChannel_05*iAlpha_Width, 3, 3); // 3 x 3 x 512 dw
fConv_9_dw = fTri_Arr(iChannel_05*iAlpha_Width, 3, 3); // 3 x 3 x 512 dw
fConv_10_dw = fTri_Arr(iChannel_05*iAlpha_Width, 3, 3); // 3 x 3 x 512 dw
fConv_11_dw = fTri_Arr(iChannel_05*iAlpha_Width, 3, 3); // 3 x 3 x 512 dw
```

```
// Memory release
funcFree_Tri_Arr(iChannel_06*iAlpha_Width, iResolution_ininital, fMain_Arr);
funcFree_Tri_Arr(iChannel_06*iAlpha_Width, iResolution_ininital, fTemp_3D_Arr);

funcFree_Tri_Arr(iChannel_01*iAlpha_Width, 3, fConv_0);
funcFree_Tri_Arr(iChannel_01*iAlpha_Width, 3, fConv_1_dw);
funcFree_Tri_Arr(iChannel_02*iAlpha_Width, 3, fConv_2_dw);
funcFree_Tri_Arr(iChannel_03*iAlpha_Width, 3, fConv_3_dw);
funcFree_Tri_Arr(iChannel_03*iAlpha_Width, 3, fConv_4_dw);
funcFree_Tri_Arr(iChannel_04*iAlpha_Width, 3, fConv_5_dw);
```

Proposed system; occupant detection

- While(1)
  - 기존 MobileNet-28의 Feed forward path와 동일하게 구성
    - Backpropagation은 불필요

```
// Main while processing
funcConv_Operation(1, iResolution_ininital, iResolution_ininital, 2, fMain_Arr, fConv_0, fTemp_3D_Arr);
funcBatch_Operation(iChannel_01*iAlpha_Width, iResolution_01, iResolution_01, fMain_Arr, fTemp_3D_Arr, fGamma_Batch, fBeta_Batch);
funcReLU_Operation(iChannel_01*iAlpha_Width, iResolution_01, iResolution_01, fMain_Arr);

//1
funcDepth_Operation(iChannel_01*iAlpha_Width, iResolution_01, iResolution_01, 1, fMain_Arr, fConv_1_dw, fTemp_3D_Arr);
funcBatch_Operation(iChannel_01*iAlpha_Width, iResolution_01, iResolution_01, fMain_Arr, fTemp_3D_Arr, fGamma_Batch, fBeta_Batch);
funcReLU_Operation(iChannel_01*iAlpha_Width, iResolution_01, iResolution_01, fMain_Arr);
fincPoint_Operation(iChannel_01*iAlpha_Width, iResolution_01, iResolution_01, iChannel_02*iAlpha_Width, fMain_Arr, fConv_1_pw, fTemp_3D_Arr);
funcBatch_Operation(iChannel_02*iAlpha_Width, iResolution_01, iResolution_01, fMain_Arr, fTemp_3D_Arr, fGamma_Batch, fBeta_Batch);
funcReLU_Operation(iChannel_02*iAlpha_Width, iResolution_01, iResolution_01, fMain_Arr);

//2
funcDepth_Operation(iChannel_02*iAlpha_Width, iResolution_01, iResolution_01, 2, fMain_Arr, fConv_2_dw, fTemp_3D_Arr);
funcBatch_Operation(iChannel_02*iAlpha_Width, iResolution_02, iResolution_02, fMain_Arr, fTemp_3D_Arr, fGamma_Batch, fBeta_Batch);
funcReLU_Operation(iChannel_02*iAlpha_Width, iResolution_02, iResolution_02, fMain_Arr);
fincPoint_Operation(iChannel_02*iAlpha_Width, iResolution_02, iResolution_02, iChannel_03*iAlpha_Width, fMain_Arr, fConv_2_pw, fTemp_3D_Arr);
funcBatch_Operation(iChannel_03*iAlpha_Width, iResolution_02, iResolution_02, fMain_Arr, fTemp_3D_Arr, fGamma_Batch, fBeta_Batch);
funcReLU_Operation(iChannel_03*iAlpha_Width, iResolution_02, iResolution_02, fMain_Arr);
```

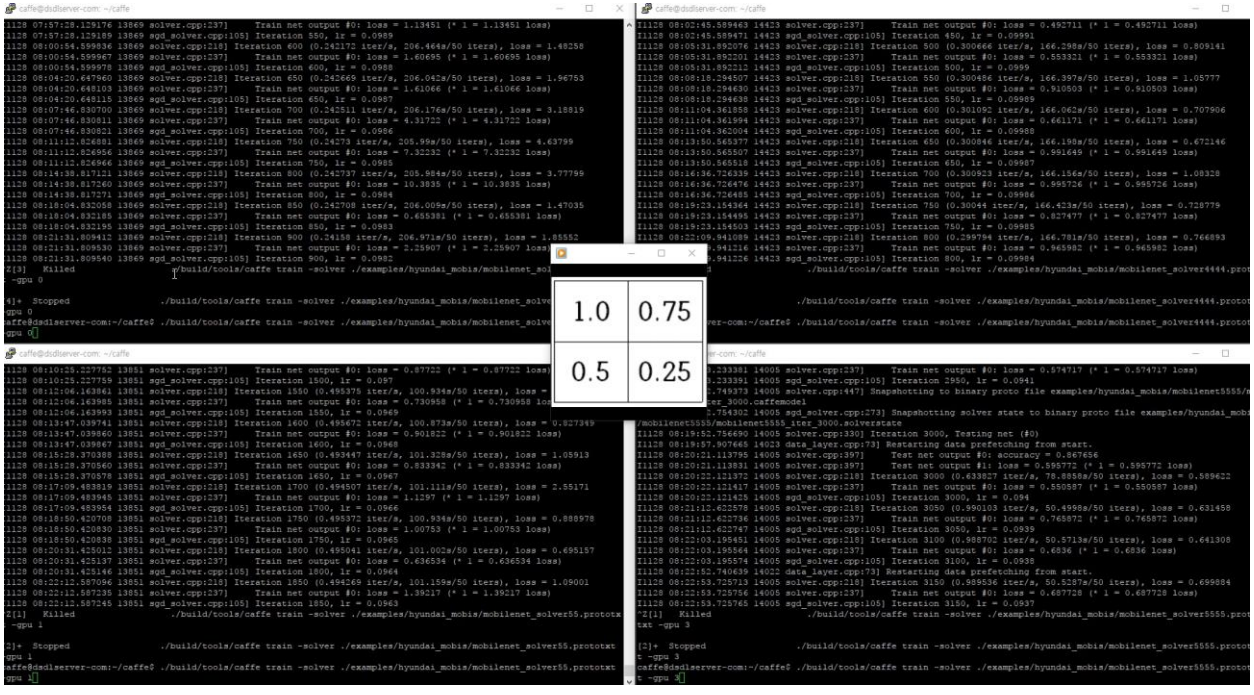
MobileNet Conv layer.

Implementation

- Caffe 기반 학습기(desktop)
  - CPU: Intel Xeon E5-2687W v4(3.00GHz, 24cores)
  - GPU: GeForce GTX 1080 Ti (250W, 11GB)
  - RAM: 128GB (Samsung DDR4 PCR-17000)
  - SW: Ubuntu 16.04, CUDA 8, cuDNN5
- Keras 기반 학습기 및 검출기(laptop)
  - CPU: Intel Core i7-6700HQ (2.60GHz)
  - RAM: 16GB
  - SW: Windows 10, Anaconda 3.0
- C 코드 기반 검출기(desktop)
  - CPU: Intel Core i5-6600 (3.30GHz)
  - RAM: 16GB (Samsung DDR4 PCR-17000)
  - SW: Windows 10, Visual studio 2015

Experimental results; MobileNet-28

- 시스템 학습 영상
  - MobileNet-28
    - 각 Layer의 Filter 수(width size)를 기준으로 하여 실행
      - Reference architecture를 “1.0”로 하여 Width size에 축소하여 네트워크 구성



Experimental results; MobileNet-28

- 시스템 학습 완료 영상
  - MobileNet-28
    - Width size 0.25 (accuracy: 0.864375, loss = 0.539687)

```
I1129 15:36:45.465075 17172 solver.cpp:310] Iteration 50000, loss = 0.572861
I1129 15:36:45.465104 17172 solver.cpp:330] Iteration 50000, Testing net (#0)
I1129 15:37:24.401593 17201 data_layer.cpp:73] Restarting data prefetching from start.
I1129 15:37:41.070935 17172 solver.cpp:397]     Test net output #0: accuracy = 0.864375
I1129 15:37:41.070972 17172 solver.cpp:397]     Test net output #1: loss = 0.539687 (* 1 = 0.539687 loss)
I1129 15:37:41.070977 17172 solver.cpp:315] Optimization Done.
I1129 15:37:41.070981 17172 caffe.cpp:259] Optimization Done.
[3]+  Killed                  ./build/tools/caffe train -solver ./examples/hyundai_mobis/vgg_solver.prototxt -gpu 1
```

    - Width size 0.5 (accuracy: 0.864375, loss = 0.577509)

```
I1129 00:30:47.680284 17189 solver.cpp:310] Iteration 50000, loss = 0.555816
I1129 00:30:47.680316 17189 solver.cpp:330] Iteration 50000, Testing net (#0)
I1129 00:31:07.244951 17209 data_layer.cpp:73] Restarting data prefetching from start.
I1129 00:31:15.610191 17189 solver.cpp:397]     Test net output #0: accuracy = 0.86375
I1129 00:31:15.610229 17189 solver.cpp:397]     Test net output #1: loss = 0.577509 (* 1 = 0.577509 loss)
I1129 00:31:15.610239 17189 solver.cpp:315] Optimization Done.
I1129 00:31:15.610244 17189 caffe.cpp:259] Optimization Done.
[7]+  Killed                  ./build/tools/caffe train -solver ./examples/hyundai_mobis/mobilenet_solver44.prototxt
-gpu 3
```

Experimental results; MobileNet-28

- MobileNet-28 architecture 변경
  - Computation cost 감소
    - Reference 논문 기반 Width 감소 (Width multiplier) 도입
      - 실험 결과를 통하여 DB 구성에 따른 Width에 강건한 시스템 구현 가능할 수도 있을 것으로 예상
        - ✓Real-time 시스템 구축 용이
  - Customized DB를 통한 Tiny network 구성
    - ImageNet에 비해 Binary 결과 출력 (1000 classes → 2 classes)
    - Tiny network에서 Pointwise convolution은 제한된 채널 수로 인한 Complexity constraint 발생 및 정확도 감소를 유발

Width multiplier		1.0 MobileNet-224	0.75 MobileNet-224	0.5 MobileNet-224	0.25 MobileNet-224
Million Mult-Adds		569	325	149	41
Million Parameters		4.2	2.6	1.3	0.5
ImageNet Accuracy		70.6%	68.4%	63.7%	50.6%
제작 DB	Accuracy	0.8648	0.8695	0.8644	0.8644
	50 iteration 당 소요 sec	203.598	165.633	101.349	50.5592

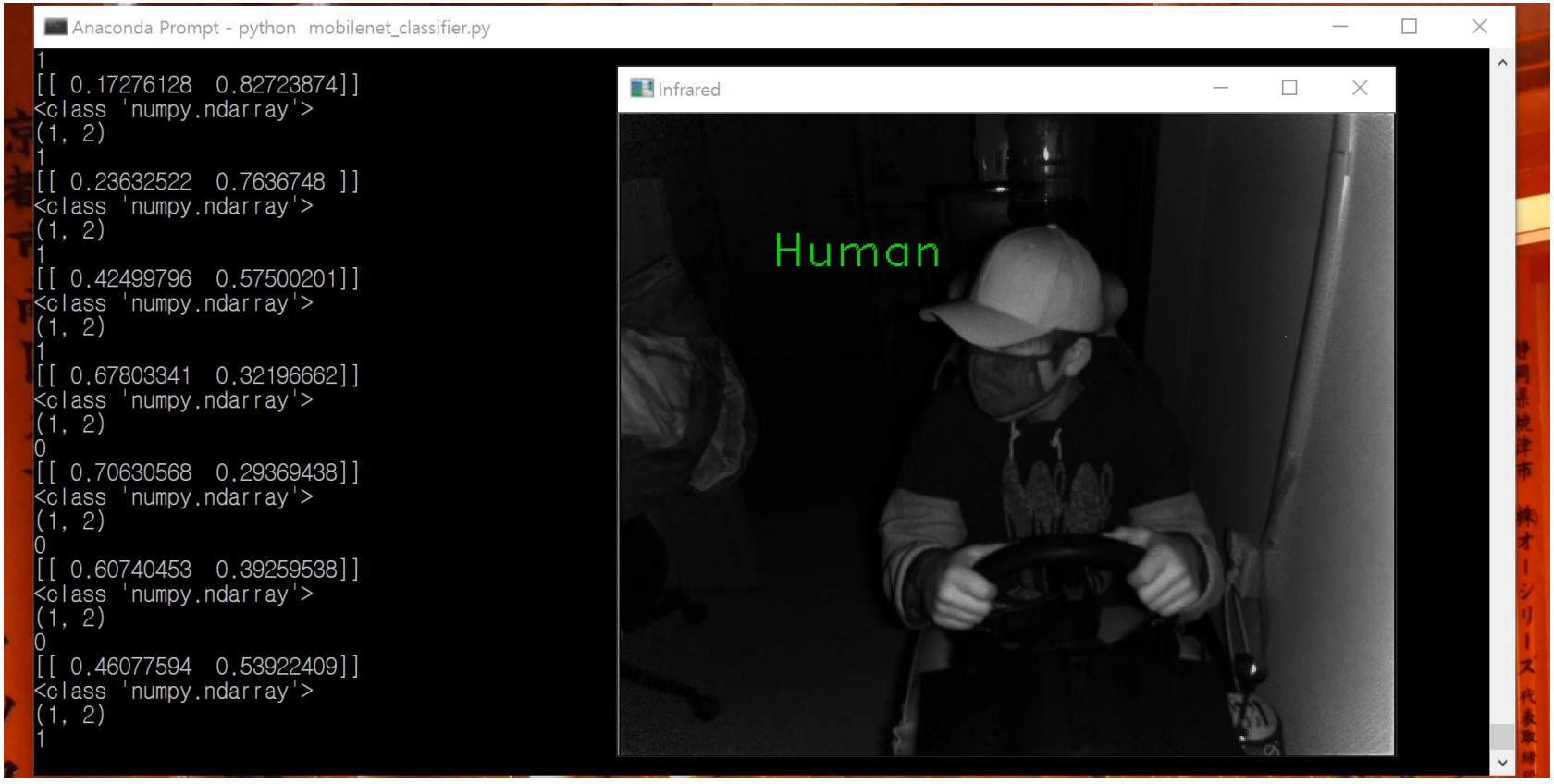
Width multiplier (total iteration: 50000)

Experimental results; occupant detection

- Python(Keras library 이용) 기반 탑승객 검출기(두꺼운 옷)

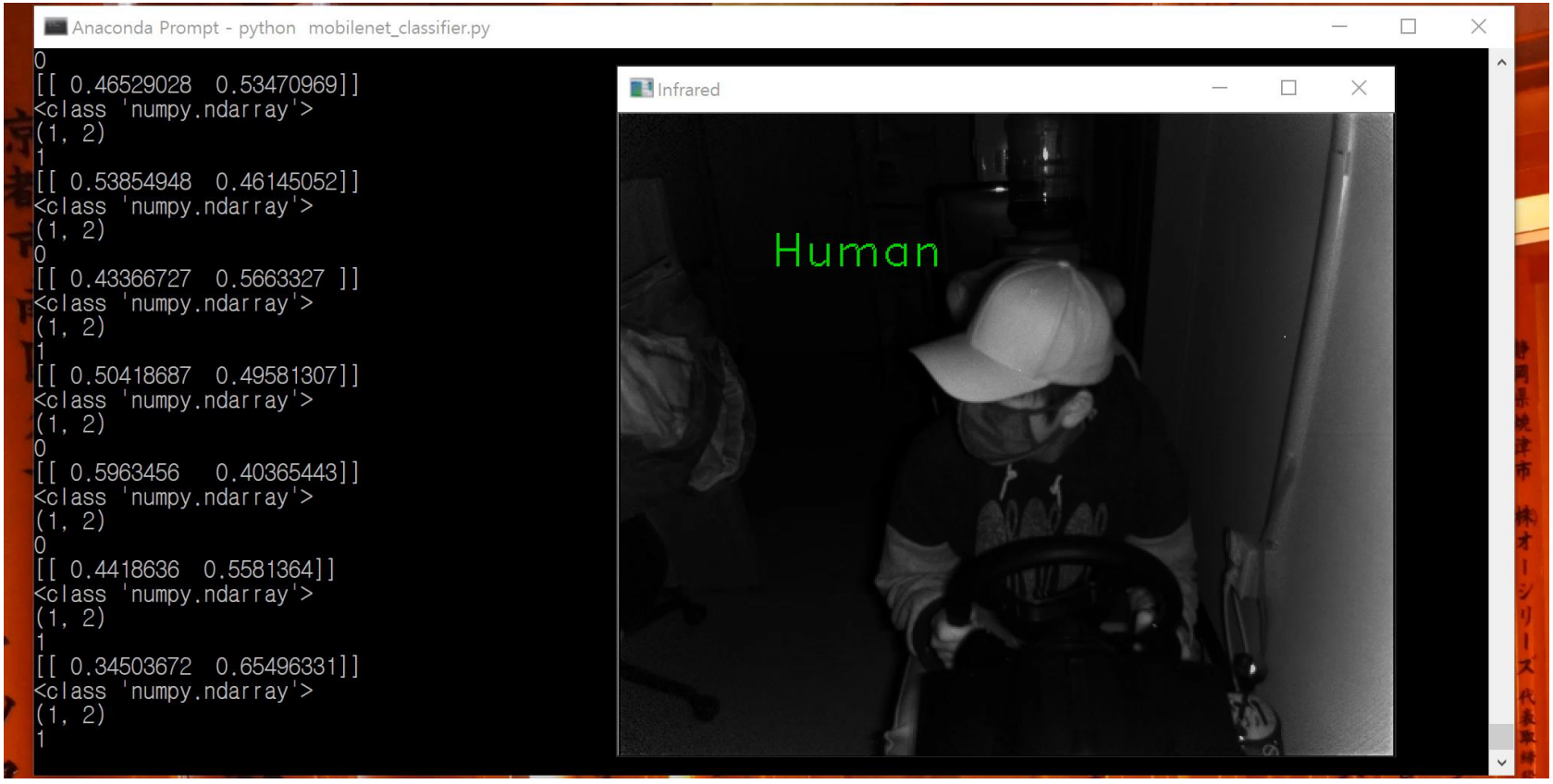
Experimental results; occupant detection

- Python(Keras library 이용) 기반 탑승객 검출기(얇은 옷)



Experimental results; occupant detection

- Python(Keras library 이용) 기반 탑승객 검출기(Error DB 촬영시)



Experimental results; occupant detection

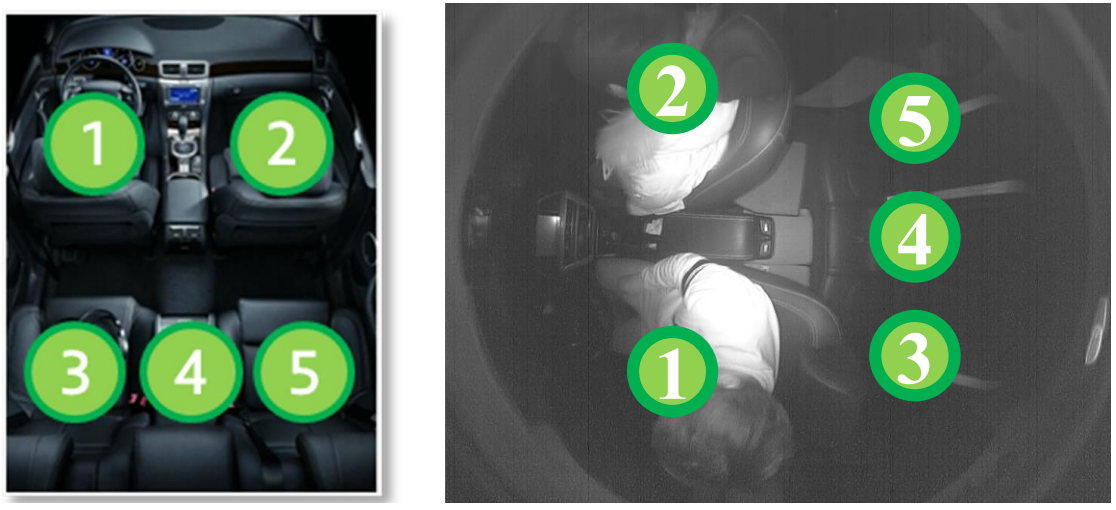
- C 기반 탑승객 검출기

Conclusion

- IR 영상을 이용한 딥러닝 학습 가능
  - Custumized IR DB를 이용한 학습 가능
    - Computer vision을 이용한 DB Augmentation실시하여 DB 양 확대 가능
  - Luminance 변화에 강건한 시스템 구현 가능
    - Night vision 카메라 보다 열화상카메라에서 더 좋은 성능 보임
- 학습기와 구현기 분리화
  - 학습된 Weight를 Load하여 사용 가능
    - 학습기에 대한 Library의 자율성 보장
  - Weight를 Load할 Layer 구성은 동일하게 구성 가능
    - Weight 구성 순서만 명확히 인지 필요
- Library 속성
  - 각 딥러닝 Library에 따른 특성 차이 인지
    - Library 마다의 장·단점 파악 후 활용 필요

Future works

- 실제 차량 환경에서 IR Database 구축 및 테스트
  - 차량내 카메라를 설치하여 영상 데이터 수집
  - 조도 측정 및 특정 시나리오 제작 등을 통하여 여러 시뮬레이션 상황 실시
- Embedded board에 최적화된 코드 구현
  - 병렬 연산을 통하여 연산 속도 향상
  - Target board에 최적화된 Library 이용



실제 차량 환경에서의 IR 카메라를 이용한 탑승객 관찰 화면

Reference

[1] K. Simonyan, et al, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” Computer Vision and Pattern Recognition, 2015.

[2] X. Zhang, et al, “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices,” Computer Vision and Pattern Recognition, 2017.

[3] I. J. Goodfellow, “Generative Adversarial Networks,” Machine Learning, 2014.

[4] A. G. Howard, et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv, 2017.

[5] K. He, et al., "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," ICCV, 2015.