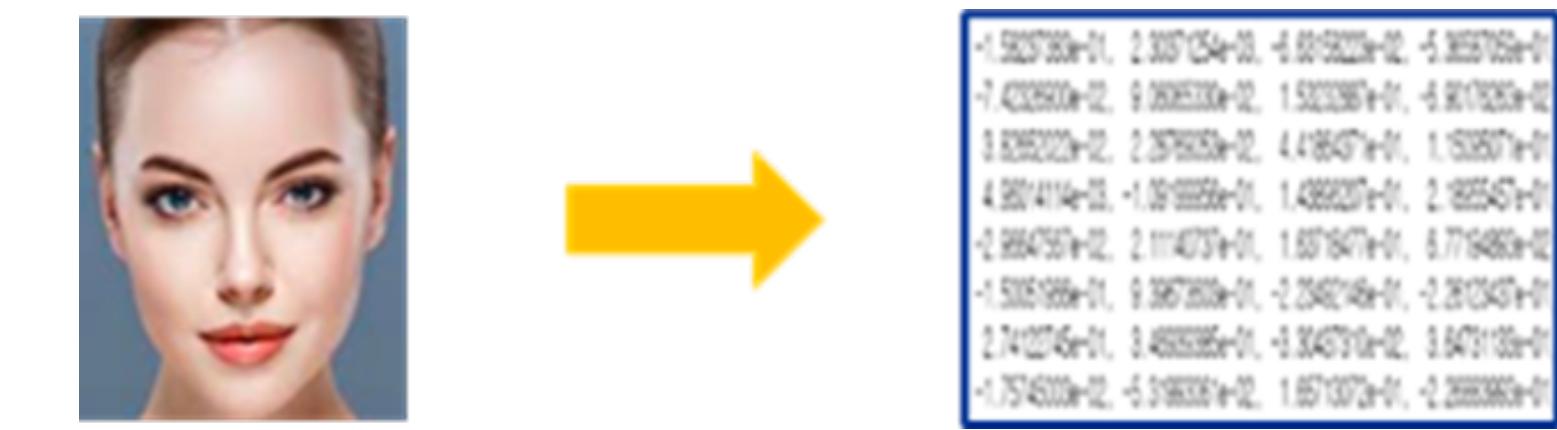
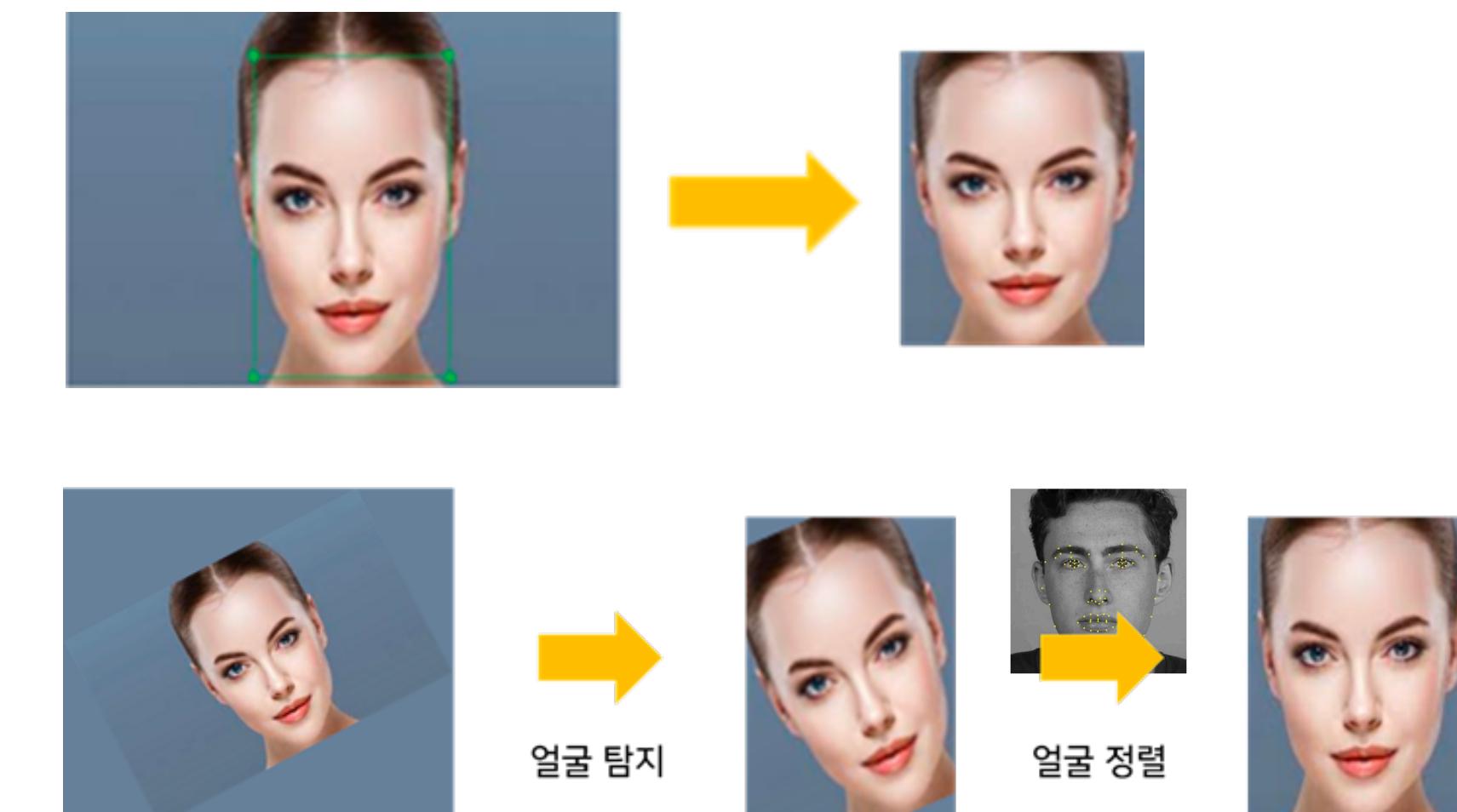
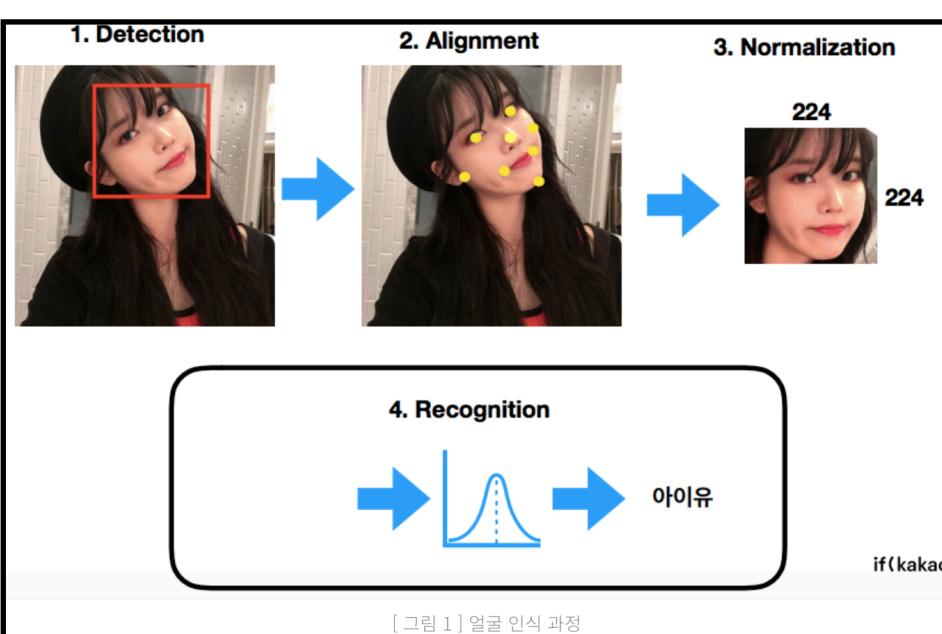


Face Detection

얼굴 인식 과정 중 첫번째 단계

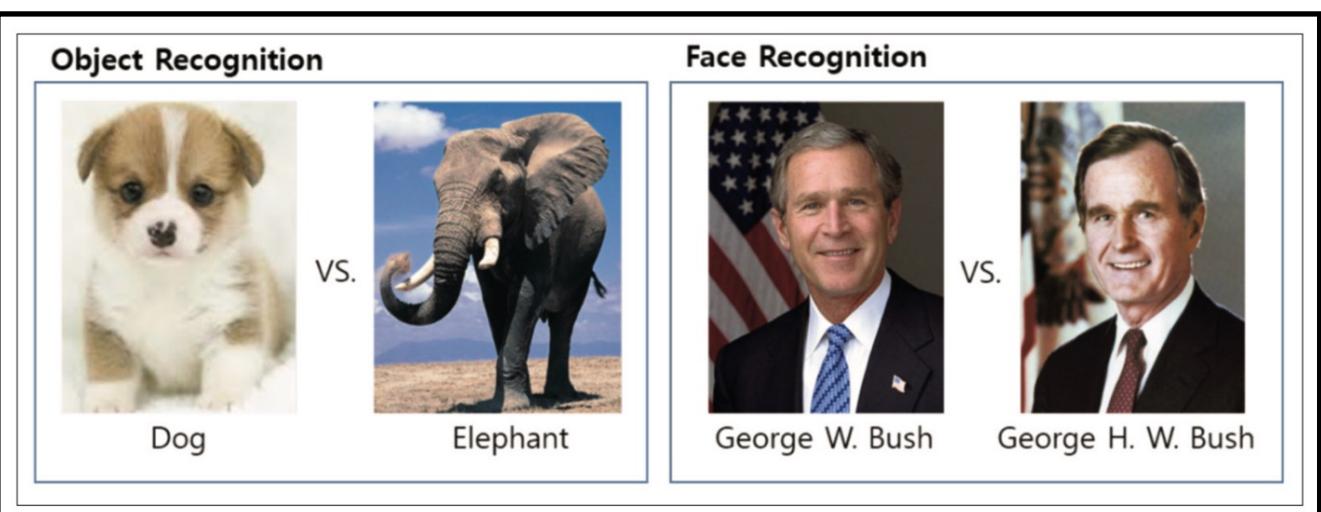
- 얼굴 인식 구조

1. 얼굴 검출 (Detection): 전체 이미지 중 얼굴 부분을 탐지하고 잘라내는 과정 (불필요한 요소 제거)
2. 얼굴 랜드마크 검출: 랜드마크 정보를 이용해 얼굴정렬, 광도보정, 영상 정규화 같은 전처리 수행 (어파인 변환)
3. 얼굴 특징 추출: 얼굴 이미지 내 특징들을 추출하여 숫자 데이터로 대응
4. 얻은 특징으로 분류모델 학습
(목적에 따라 달라질 수 있음)

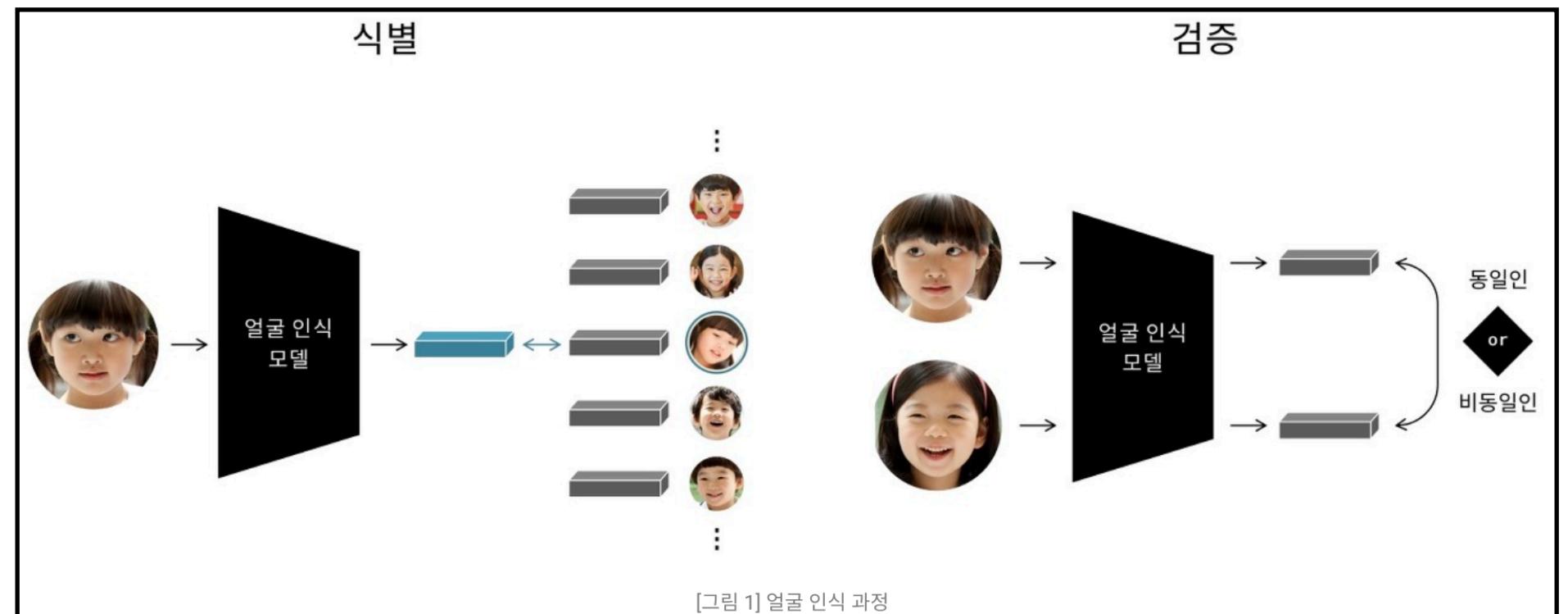


Face Detection

- 얼굴 인식 (Face Recognition)
 - 물체 인식의 하나의 종류 (Object 인식 vs face 인식)
 - 얼굴 인식에 특화된 딥러닝 아키텍처 ↑
 - 얼굴 인식의 난이도가 훨씬 높음
 - 동일 Object의 차이를 인식해야 함
(유사한 패턴 내에서 identity를 구별해야 함)
 - 조명 변화, Occlusion - Big Issue
 - 개별 모듈도 수요가 큼 (얼굴 검출 모듈로 카메라 포커스 맞추기)

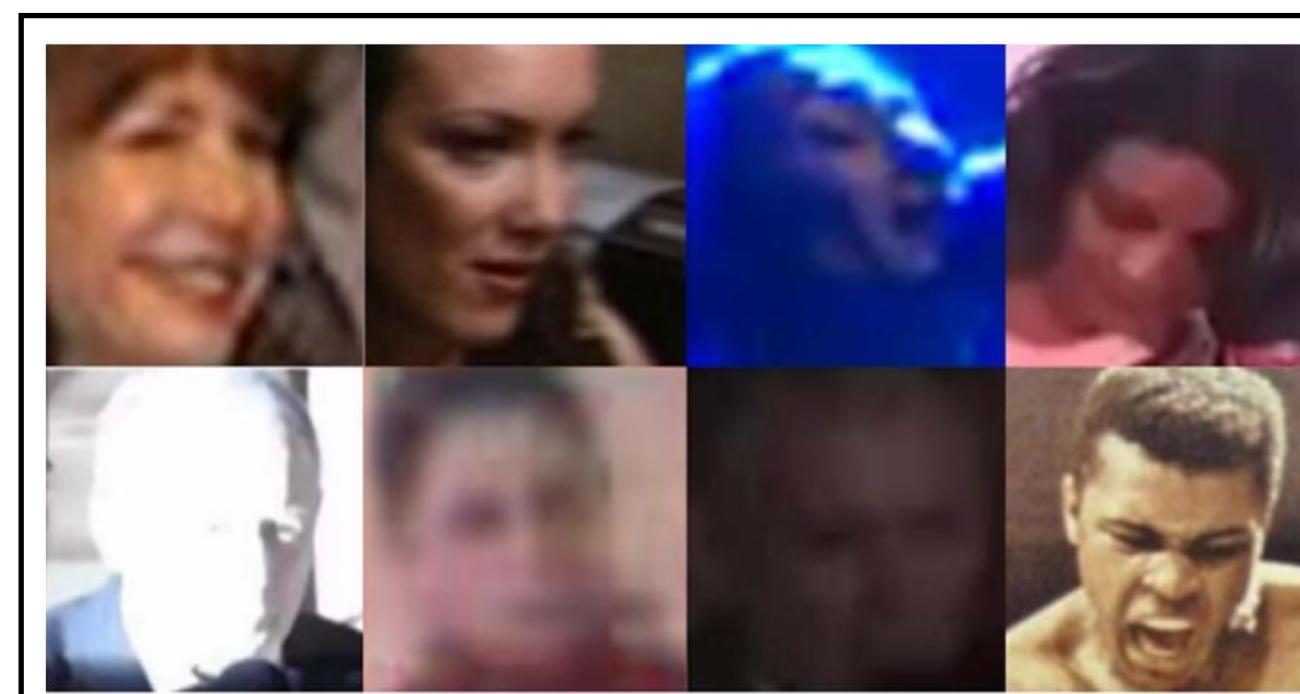


- 종류
 - Face Verification : Face Identification = (1 : 1) : (1 : N)



- 실제 환경에서 취득되는 얼굴 이미지의 어려움

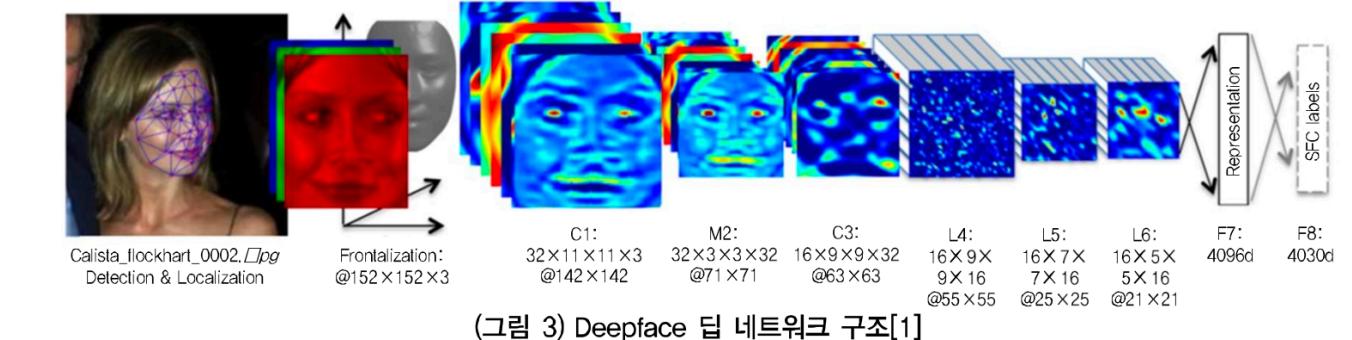
- ! 포즈 변화(Orientation) 및 조명(Lighthing) 변화
- 원거리 촬영과 사람의 움직임에 의한 저해상도 / blur
- Occlusion, Accessories, Expression
- 나이, 성별, 인종 간 인식 성능 차이



Face Detection

트렌드 1. 얼굴인식 딥 네트워크 구조에 대한 연구

- DeepFace (2014, CVPR, FaceBook)
 - 얼굴 인식에서 딥러닝이 처음으로 접목된 연구
 - Facebook이 내부적으로 수집한 대용량의 데이터 이용 (당시 공개 X)
 - 네트워크 구조
 - 사전에 학습된 3D 얼굴 기하 모델을 이용하여 랜드마크 추출
 - 추출 후 어파인(affine) 변환에 의해 얼굴 정렬을 수행
 - 9개의 층으로 구성된 컨볼루션(Convolution) 신경망 학습
- VGGFace (2015)
 - 인터넷 검색을 통해 직접 만든 대용량의 얼굴인식을 위한 데이터셋인 **VGG 얼굴 데이터셋을 공개 (얼굴 인식 가능한지 확인)**
 - 15개의 컨볼루션 층으로 구성된 딥러닝 네트워크 구조를 학습
 - VGG 구조와 마찬가지로 상대적으로 간단한 3×3 Conv 필터를 이용
 - VGGFace는 LFW 데이터셋 (**얼굴 인식 가능한지 확인**)에 대해 DeepFace보다 약 1% 정도 성능 개선 (98.95%)

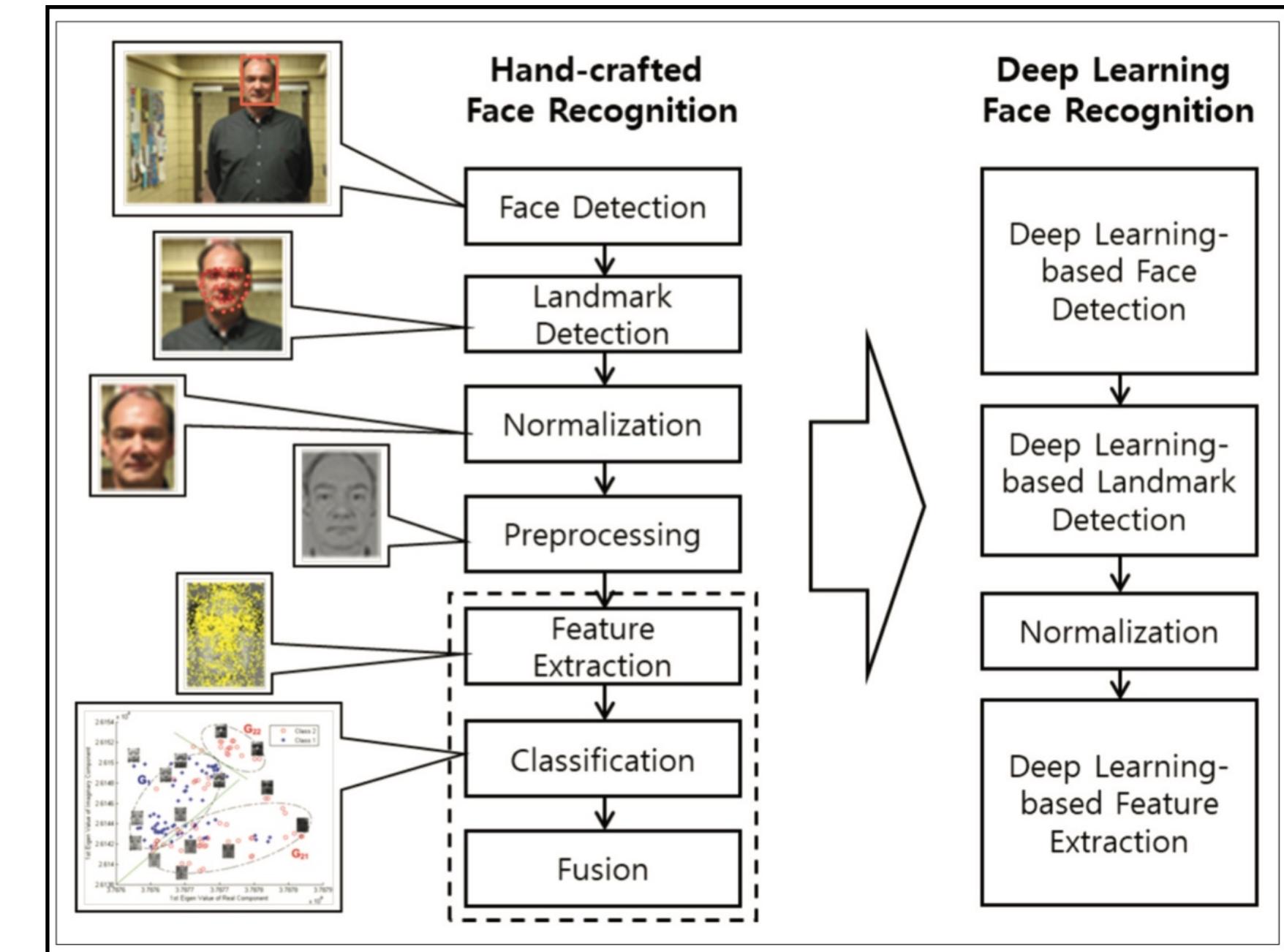


(그림 3) Deepface 딥 네트워크 구조[1]



(그림 4) VGGFace 딥 네트워크 구조

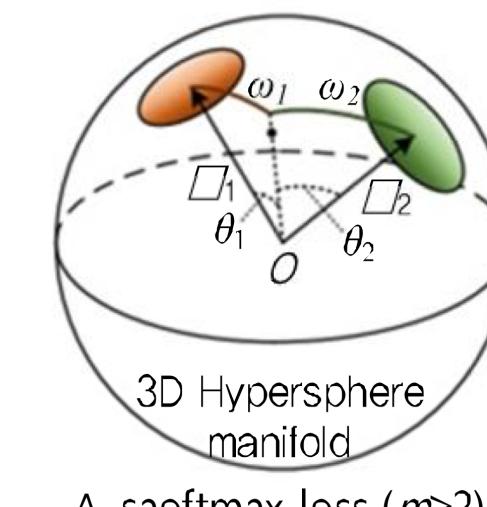
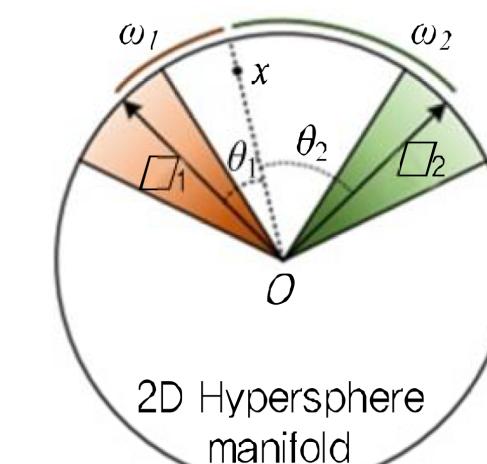
- 이후, DeepID, DeepID2, DeepID2+, DeepID3와 같은 얼굴인식을 위한 다양한 딥러닝 네트워크 구조가 제안됨



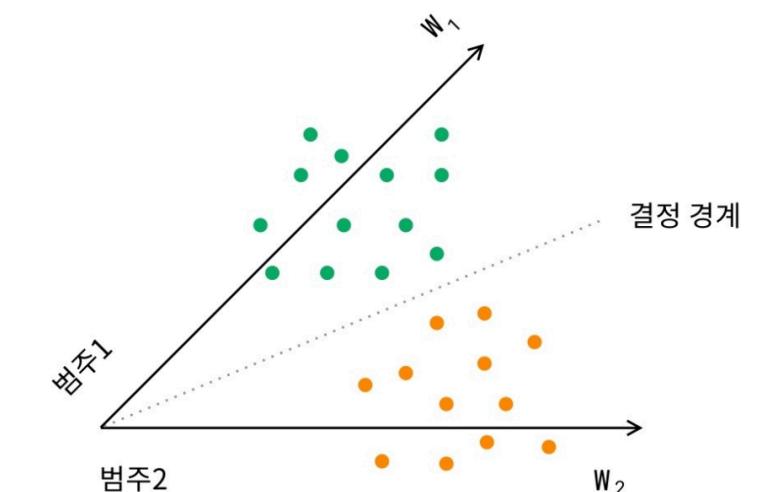
Face Detection

트렌드 2. 손실함수(Loss Function)의 재정의

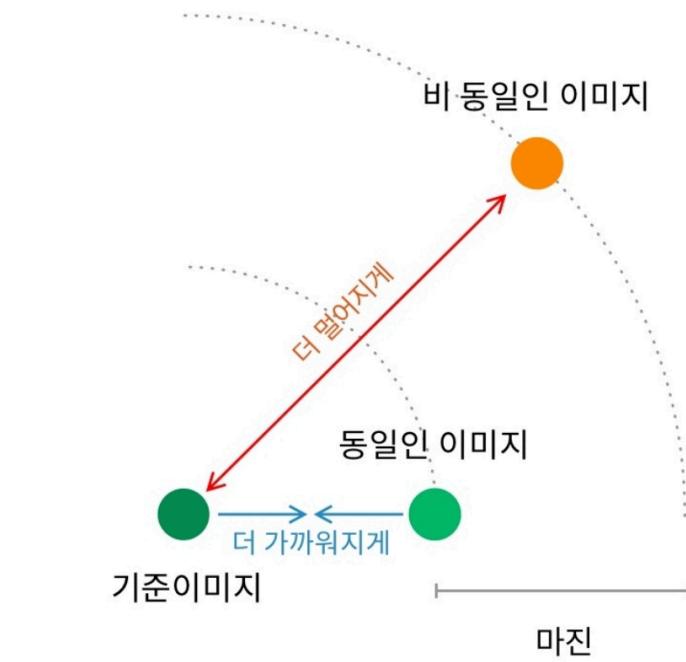
- 분별력 있는 특징을 학습을 위해 거리 척도 학습(Distance Metric Learning)에 대한 연구가 수행
- 주요 목적
 - 동일 인물로부터 추출된 특징의 분산은 작게 하고, 다른 인물로부터 추출된 특징의 분산은 크게 하는 것
 - 개체 간 분별(Interclass Separability)이 잘 되어야 함
 - 동일 개체 간 긴밀성(Intraclass Compactness)이 높아야 함
- 일반적인 분류 모델과의 차이와 손실함수의 재정의
 - 일반 모델: 교차 엔트로피 손실 함수(Cross entropy Loss)와 같은 소프트맥스(Softmax) 기반 손실 함수를 학습에 사용
 - 소프트맥스 기반 손실 함수는 동일 개체 간 긴밀성을 전혀 고려하지 않은 손실 함수
 - 동일 개체 간 긴밀성을 높이기 위한 학습
 - Center Loss, Triplet Loss와 같이 거리 기반의 손실 함수를 사용하여 학습 (Metric Learning)
 - Cosface, Arcface와 같이 기존 Softmax에 각도 개념을 도입한 손실 함수를 사용하여 학습(Angular Margin Learning)
- 예시
 - 구글에서 발표한 FaceNet - Triplet Loss 정의하여 딥러닝 학습
 - 동일한 인물에 대해 추출된 특징들 사이의 유클리디언(Euclidean) 거리가 다른 인물들로부터 추출된 특징들 사이의 유클리디언 거리보다 작다는 Triplet Loss를 정의 threshold
 - SphereFace - A-Softmax 정의하여 딥러닝 학습
 - hypersphere 공간에서의 고차원의 영상 데이터 맵핑(mapping)을 위한 angular softmax(A-Softmax) 손실함수를 정의
 - CosFace - LMCL 정의하여 딥러닝 학습
 - Large Margin Cosine(LMCL) 손실함수를 정의
 - 그 외에도 Ring Loss, ArcFace 등의 연구를 수행



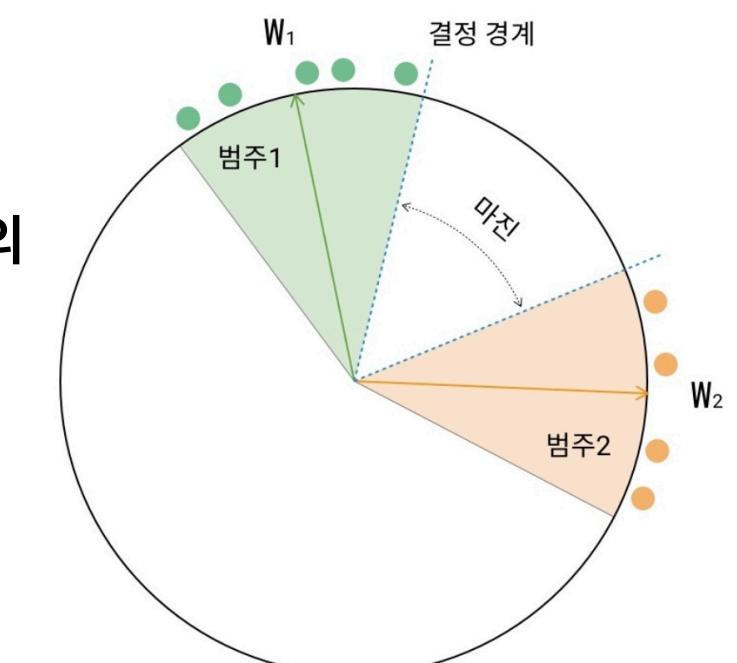
A-softmax loss ($m \geq 2$)



[그림 4] 소프트맥스 손실 함수



[그림 5] 거리 기반 손실 함수



[그림 6] 앵글러 마진 기반 손실 함수인 AdaM-소프트맥스

사례 정리 1

논문명: 마스크를 착용한 얼굴 인식을 위한 방법 연구

- 2020-12월, KCI 등재

- 선글라스, 마스크 등을 착용해서 얼굴의 일부가 가려지는 Occlusion 문제 발생
 - 미세먼지나 COVID-19의 여파로 마스크 착용하는 경우가 많았음
- 마스크를 착용한 경우의 인식률을 높이기 위한 방법과 특정 그룹에서 얼굴 인식을 위한 매칭 알고리즘 제안
- 목표: 마스크를 착용하지 않은 경우, 정확도 유지 / 마스크를 착용한 경우, 정확도 향상

- RetinaFace

- Multi-Task Learning: 하나의 신경망에서 여러 작업을 동시에 수행할 수 있도록 훈련
- 얼굴 탐지에서는 얼굴의 위치, 위치에 대한 신뢰도, 얼굴이 가지는 특징점 (눈, 코, 입의 양 끝을 나타내는 5개의 랜드마크), 얼굴의 3D 위치 및 대응관계까지 예측

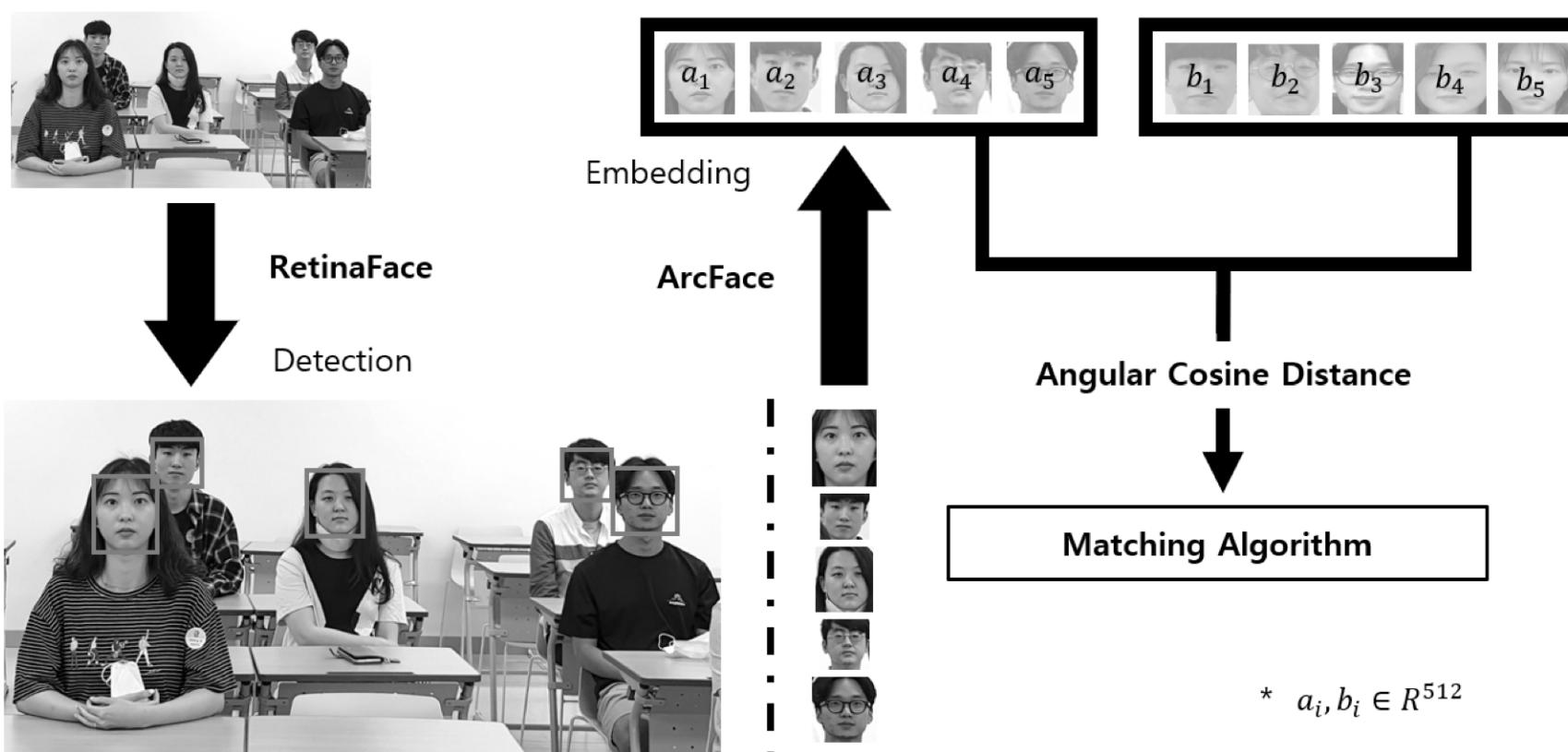
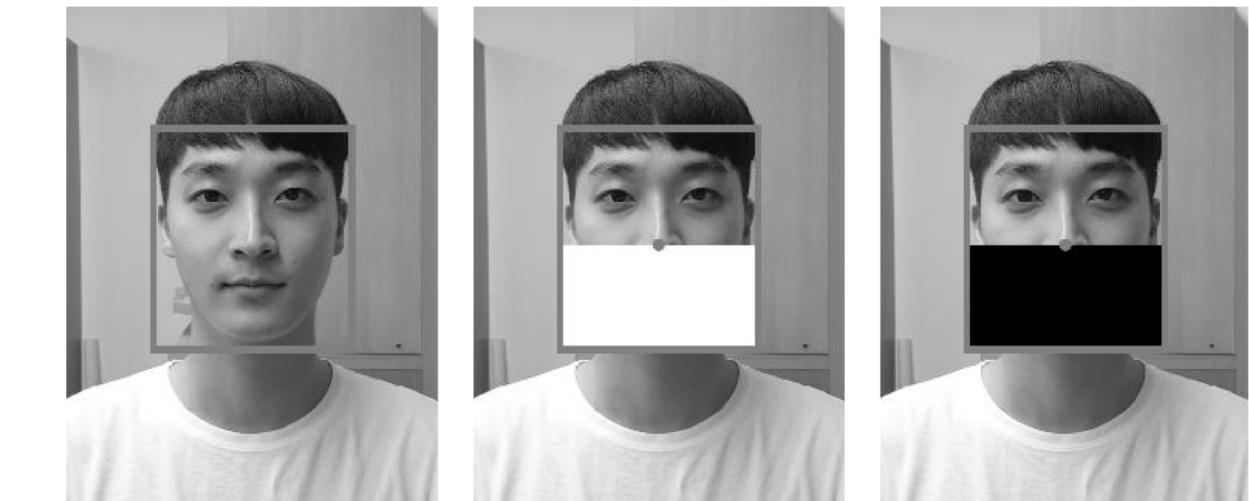


그림 1. 얼굴 인식 시스템



(a) 원본 얼굴 (b) 흰색 모자이크 (c) 검정 모자이크
그림 2. 거리를 측정할 3가지 얼굴 사진

- ArcFace

- Angular Cosine Distance로 임베딩한 두 벡터의 사이를 측정

$$ACD = \frac{\arccos(Cosine \sim ilarity)}{\pi} \quad \dots (1)$$

$$\text{where, } Cosine \sim ilarity = \frac{A \cdot B}{\|A\| \|B\|}$$

- 얼굴 하단부가 가려지는 폐색 문제 발생 - 얼굴 인식의 정확도가 낮아짐

- 폐색이 일어난 얼굴의 하단부에 모자이크를 씌워, CNN에서 하단부의 특징을 없애고 상단부의 특징만 사용하게 함
- 얼굴 이미지와 모자이크를 씌워 생성한 이미지에서의 유사성(ACD)을 측정하여 결합하는 방법을 제안

- RetinaFace가 예측하는 코의 위치를 기준으로 얼굴의 하단부를 흰색, 검정색으로 바꾼 2가지 사진을 추가로 임베딩하여 각각 거리를 측정

- 3개의 이미지에서 바운딩 박스를 측정, 측정한 거리를 가중치합으로 결합하는 방법을 제안

$$Distance = \alpha \times F + \beta \times W + \gamma \times B \quad \dots (2)$$

- 결과: 마스크를 착용하지 않은 경우 98.8 -> 97.6

마스크를 착용한 경우, 75.6 -> 95.2

사례 정리 2

YOLOv4를 이용한 얼굴검출

- 2021, Amazon AWS
- Yolo v4 vs RetinaFace 비교

- RetinaFace

추가학습과 자가지도 다중작업 학습(extra-supervised and self-supervised multi-task learning) 기법을 이용
-> 밀집된 군중에서 작은 크기의 얼굴도 검출

- YOLOv4 기반 얼굴검출 모델

전면과 측면 얼굴을 비롯하여 다양한 각도와 크기의 얼굴을 모두 검출. 하지만, 배경의 작은 얼굴들은 검출하지 못하는 경우가 다수 발견

- 얼굴이 가까이 겹친 사진

- RetinaFace

- False 얼굴 영역을 생성하는 현상 발견. 그림 2에서 RetinaFace는 두 얼굴 사이에 존재하지 않는 얼굴 bounding box를 표시함.
얼굴이 카메라와 근접해서 촬영된 사진에 대해서 검출을 하지 못하는 경우가 있음

- YOLOv4

- occlusion이 심한 상황의 얼굴도 RetinaFace에 비해 잘 검출.

- 실사 사진 외에도 사람의 얼굴 형상을 한 묘사 이미지에서도 검출. (RetinaFace 일부 검출하지만, Yolov4의 검출 빈도가 더 높음)

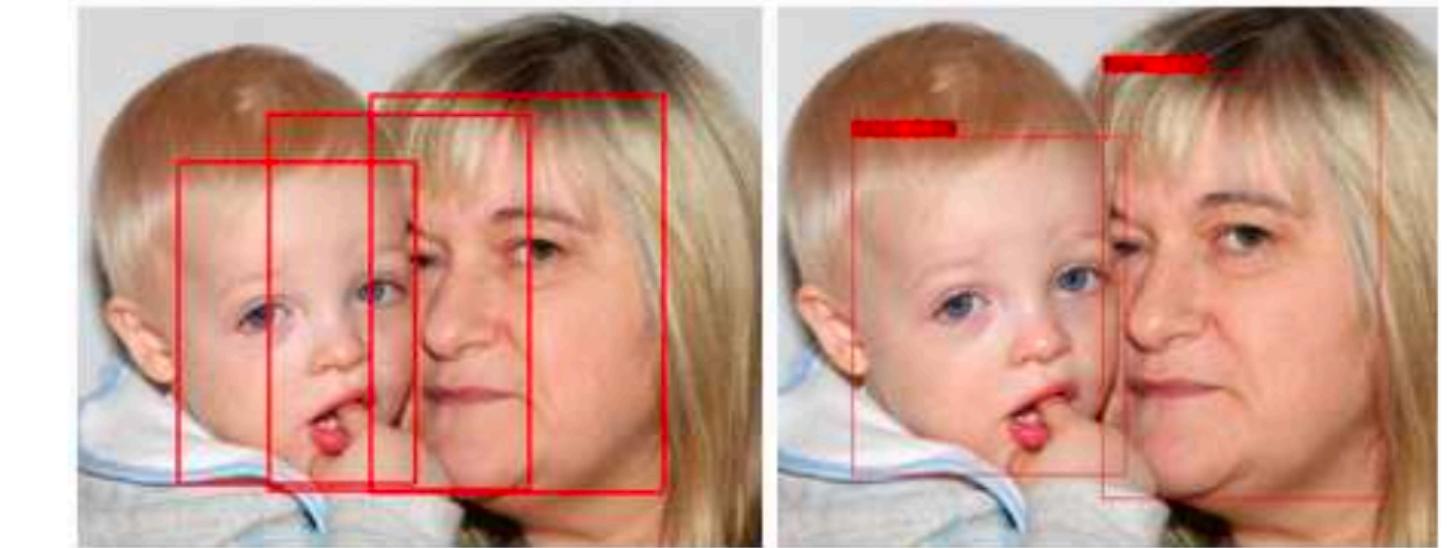
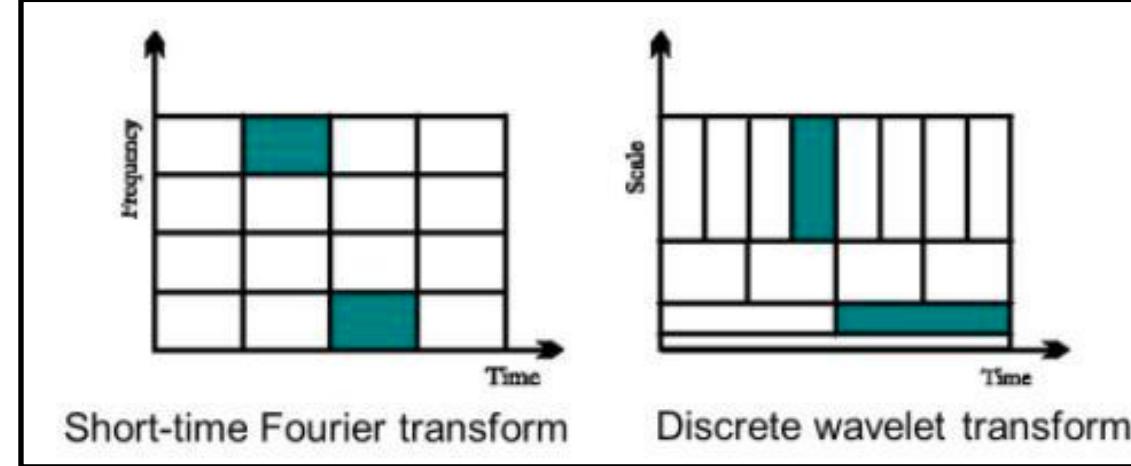


그림 2. 겹친 얼굴검출 결과 (좌) RetinaFace, (우) YOLOv4

사례 정리 3

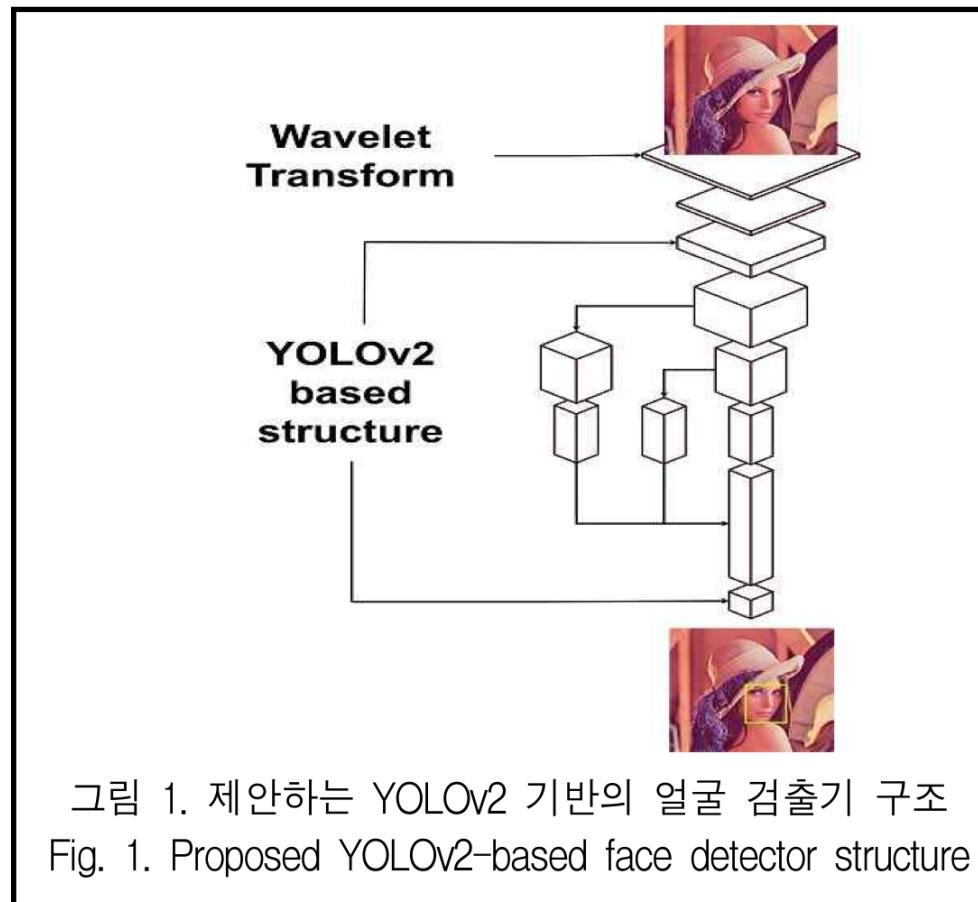
YOLOv2 구조 기반 얼굴 검출기의 성능 향상을 위한 웨이블릿 전처리 방법



- 2021, 한국정보기술학회

- Yolov2 선택

- Yolov3는 처리속도가 비교적 느려서 임베디드 시스템과 같은 프로세서 성능이 제한된 환경에서 실시간 얼굴 검출하기 힘들다고 판단
 - Yolov2의 경우, 처리 속도가 훨씬 빠르기 때문에, Yolov2 + 입력 영상에 대한 전처리로 웨이블릿 변환을 적용하여 AP를 향상시키는 것을 목표로 함



- (결과) 기존 Yolov에 비해 1.87%p 높은 정밀도와 10.99%의 높은 처리 속도를 보여줌.

- 웨이블릿 변환이란? (필요하다면 자세하게 공부)

- 푸리에 변환과 관련
 - 국부적으로 존재하는 작은 웨이블릿을 패턴을 이동 및 스케일링을 통해 임의의 파형으로 표현이 가능 + 푸리에 변환과 달리 시간축 방향으로 이동하여 시간에 대한 신호를 주파수 성분으로 분해하는 작업 (갑작스러운 변화에 강함)
 - 논문에서 크게 언급하지 않았지만, 가우시안 필터를 사용하는 경우 도 고려해봐도 좋을 듯

표 1. YOLOv2 기반의 얼굴 검출기 구조에서 제안하는 방법과 기존 방법들의 성능 비교

Table 1. Performance comparison between the proposed method and the traditional methods in the YOLOv2-based face detector structure

	평균 정밀도	처리 속도 (s)
YOLOv2 (기본 모델)	0.7838	5.807
YOLOv2 (가우시안 필터)	0.8180	2.785
제안하는 방법 (웨이블릿 변환)	0.8367	2.479

YOLO를 이용한 Face Detector (다음주 리뷰예정 - 둘 다 Yolov5)

• YOLO5FACE: Why Reinventing a Face Detector

YOLO5Face: Why Reinventing a Face Detector

Delong Qi, Weijun Tan*, Qi Yao, Jingfeng Liu
Shenzhen Deepcam Information Technologies
Shenzhen, China
{delong.qi.weijun.tan,qi.yao,jingfeng.liu}@deepcam.com
*LinkSprite Technologies, USA, weijun.tan@linksprite.com

Abstract—Tremendous progress has been made on face detection in recent years using convolutional neural networks. While many face detectors use designs designated for the detection of face, we treat face detection as a general object detection task. We implement a face detector based on YOLOv5 object detector and call it YOLO5Face. We add a five-point landmark regression head into it and use the Wing loss function. We design detectors with different model sizes, from a large model to achieve the best performance, to a super small model for real-time detection on an embedded or mobile device. Experiment results on the WiderFace dataset show that our face detectors can achieve state-of-the-art performance in almost all the Easy, Medium, and Hard subsets, exceeding the more complex designated face detectors. The code is available at <https://www.github.com/deepcam-cn/yolov5-face>.

Index Terms—Face detection, convolutional neural network, YOLO, real-time, embedded device, object detection

I. INTRODUCTION

Face detection is a very important computer vision task. Tremendous progresses have been made since deep learning, particularly convolutional neural network (CNN), has been used in this task. As the first step of many tasks, including face recognition, verification, tracking, alignment, expression analysis, face detection attracts many researches and developments in the academia and the industry. And the performance of face detection has improved significantly over the years. For a survey of the face detection, please refer to the benchmark results [1], [2]. There are many methods in this field from different perspectives. Research directions include design of CNN network, loss functions, data augmentations, and training strategies. For example, in the YOLOv4 paper, the authors explore all these research directions and propose the YOLOv4 object detector based on optimizations of network architecture, selection of bags of freebies, and selection of bags of specials [3].

In our approach, we treat the face detection as a general object detection task. We have the same intuition as the TinaFace [4]. Intuitively, face is an object. As discussed in the TinaFace [4], from the perspective of data, the properties that faces has, like pose, scale, occlusion, illumination, blur and etc., also exist in other objects. The unique properties in faces like expression and makeup can also correspond to distortion and color in objects. Landmarks are special to face, but they are not unique either. They are just key points of an object. For example, in license plate detection, landmarks are also used. And adding landmark regression in the object prediction head is straightforward. Then from the perspective

of challenges encountered by face detection like multi-scale, small faces and dense scenes, they all exist in generic object detection. Thus, face detection is just a sub task of general object detection.

In this paper, we follow this intuition and design a face detector based on the YOLOv5 object detector [5]. We modify the design for face detection considering large faces, small faces, landmark supervision, for different complexities and applications. Our goal is to provide a portfolio of models for different applications, from very complex ones to get the best performance to very simple ones to get the best trade-off of performance and speed on embedded or mobile devices.

Our main contributions are summarized as following,

- We redesign the YOLOv5 object detector [5] as a face detector, and call it YOLO5Face. We implement key modifications to the network to improve the performance in terms of mean average precision (mAP) and speed. The details of these modifications will be presented in Section III.
- We design a series of models of different model sizes, from large models, to medium models, to super small models, for needs in different applications. In addition to the backbone used in YOLOv5 [5], we implement a backbone based on ShuffleNetV2 [6], which gives the state-of-the-art (SOTA) performance and fast speed for mobile device.
- We evaluate our models on the WiderFace [1] dataset. On VGA resolution images, almost all our models achieve the SOTA performance and fast speed. This proves our goal, as the title of this paper claims, we do not need to reinvent a face detector since the YOLO5Face can accomplish it.

II. RELATED WORK

A. Object Detection

General object detection aims at locating and classifying the pre-defined objects in a given image. Before deep CNN is used, traditional face detection uses hand crafted features, like HAAR, HOG, LBP, SIFT, DPM, ACF, etc. The seminal work by Viola and Jones [7] introduces integral image to compute HAAR-like features. For a survey of face detection using hand crafted features, please refer to [8], [9].

Since the deep CNN shows its power in many machine learning tasks, face detection is dominated by deep CNN

• YOLO-FaceV2: A Scale and Occlusion Aware Face Detector

YOLO-FaceV2: A Scale and Occlusion Aware Face Detector

Ziping Yu¹, Hongbo Huang^{*2}, Weijun Chen³, Yongxin Su⁴, Yahui Liu⁵, and Xiuying Wang²

¹School of Instrument Science and Opto-electronic Engineering, Beijing Information Science and Technology University, Beijing, China

²Computer School, Beijing Information Science and Technology University, Beijing, China

³Data Algorithm, NIO, Shanghai, China

⁴School of Mechanical and Electrical Engineering, Beijing Information Science and Technology University, Beijing, China

⁵School of Information Management, Beijing Information Science and Technology University, Beijing, China

Abstract

In recent years, face detection algorithms based on deep learning have made great progress. These algorithms can be generally divided into two categories, i.e. two-stage detector like Faster R-CNN and one-stage detector like YOLO. Because of the better balance between accuracy and speed, one-stage detectors have been widely used in many applications. In this paper, we propose a real-time face detector based on the one-stage detector YOLOv5, named YOLO-FaceV2. We design a Receptive Field Enhancement module called RFE to enhance receptive field of small face, and use NWD Loss to make up for the sensitivity of IoU to the location deviation of tiny objects. For face occlusion, we present an attention module named SEAM and introduce Repulsion Loss to solve it. Moreover, we use a weight function Slide to solve the imbalance between easy and hard samples and use the information of the effective receptive field to design the anchor. The experimental results on WiderFace dataset show that our face detector outperforms YOLO and its variants can be found in all easy, medium and hard subsets. Source code in <https://github.com/Krasjet-Yu/YOLO-FaceV2>

Keywords— Face detection, YOLO, Scale-Aware, Loss function, Imbalance problem

1 Introduction

Face detection is an essential step in many face-related applications, such as face recognition, face verification and face attribute analysis, etc. With the booming of deep convolutional neural networks in recent years, the performance of face detectors has been greatly improved. Many high-performance face detection algorithms based on deep learning have been proposed. Generally, these algorithms can be divided into two branches. One branch of typical deep-learning-based face detection algorithms [1, 2, 3] uses cascading means of neural networks as

*Corresponding Author: hhb@bistu.edu.cn

• KAKAO - GroupFace

YOLO - 실습

- **Darknet의 경우, 대표적으로 두 가지 버전이 있음**
 1. pjreddie, 2. AlexeyAB
- **AlexeyAB**
 - pjreddie git page의 contributor로 되어 있음
 - pjreddie 버전을 사용할 경우, 깃허브 원본의 경우, cuda 8 버전은 안 됨.
-> 다른 사람이 수정한 코드를 가져오거나 Docker 환경에서 새로 설치해야함
 - AlexeyAB는 cuda 8 버전을 설치하도록 명시되어 있음
-> 저번주에 설치한 환경으로 AlexeyAB 버전으로 돌려본 결과, 잘 돌아갔음을 확인
- **(실행 사진 추가 - 뒷장)**

Scaled-YOLOv4:

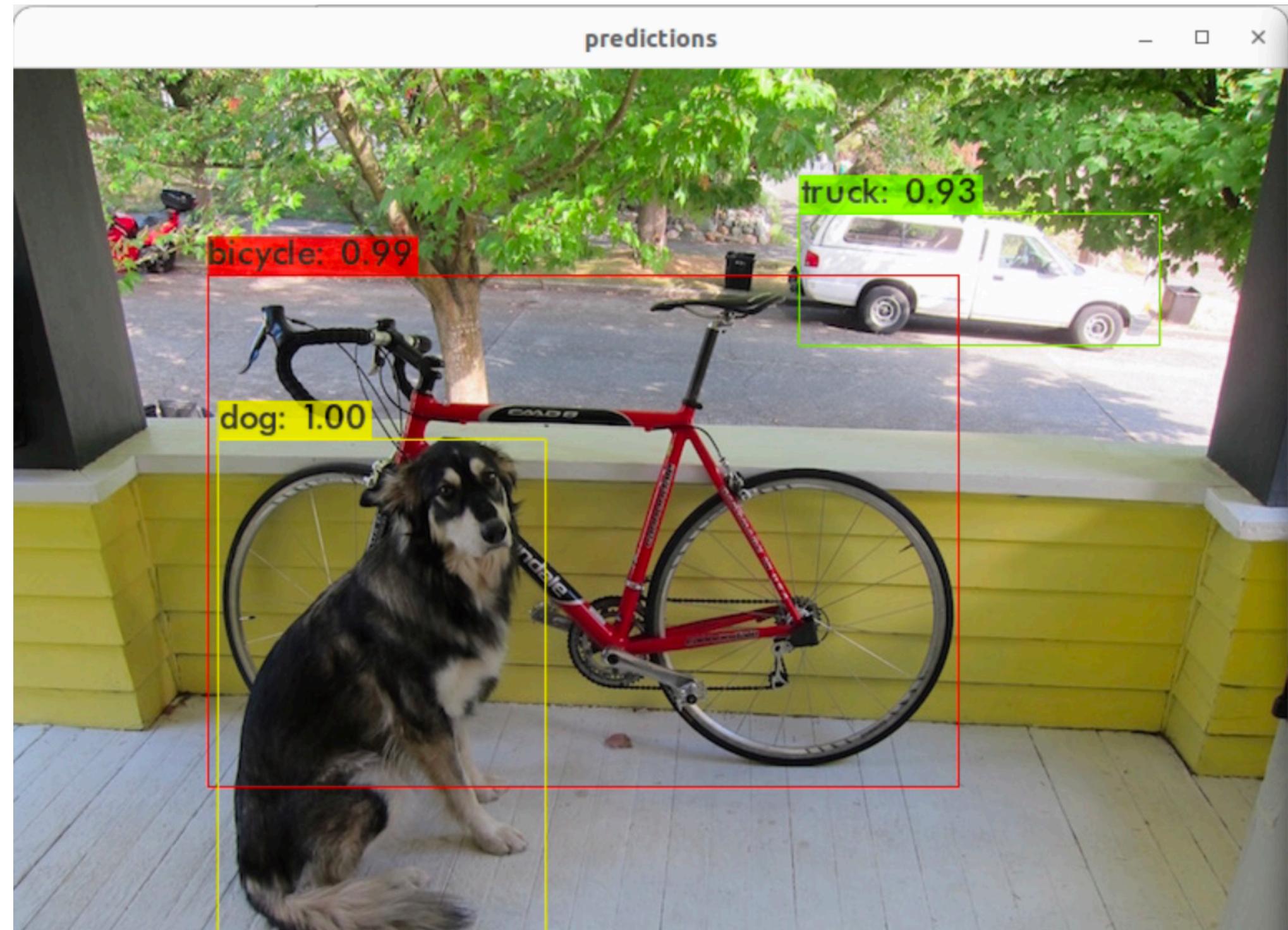
- paper (CVPR 2021): https://openaccess.thecvf.com/content/CVPR2021/html/Wang_Scaled-YOLOv4_Scaling_Cross_Stage_Partial_Network_CVPR_2021_paper.html
- source code - Pytorch (use to reproduce results): <https://github.com/WongKinYiu/ScaledYOLOv4>
- source code - Darknet: <https://github.com/AlexeyAB/darknet>
- Medium: https://alexeyab84.medium.com/scaled-yolo-v4-is-the-best-neural-network-for-object-detection-on-ms-coco-dataset-39dfa22fa982?source=friends_link&sk=c8553bfed861b1a7932f739d26f487c8

YOLOv4:

- paper: <https://arxiv.org/abs/2004.10934>
- source code: <https://github.com/AlexeyAB/darknet>
- Wiki: <https://github.com/AlexeyAB/darknet/wiki>
- useful links: https://medium.com/@alexeyab84/yolov4-the-most-accurate-real-time-neural-network-on-ms-coco-dataset-73adfd3602fe?source=friends_link&sk=6039748846bbcf1d960c3061542591d7

For more information see the [Darknet project website](#).

pjreddie 깃허브 Readme



```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

./darknet detect cfg/yolov3.cfg yo... x user@user-workstation3090:~/De... x user@user-wor

s data/dog.jpg

CUDA-version: 11070 (11070), cuDNN: 8.6.0, CUDNN_HALF=1, GPU count: 1
CUDNN_HALF=1
OpenCV version: 4.5.4
0 : compute_capability = 860, cudnn_half = 1, GPU: NVIDIA GeForce RTX 3090
net.optimized_memory = 0
mini_batch = 1, batch = 1, time_steps = 1, train = 0
layer filters size/strd(dil) input output
0 Create CUDA-stream - 0
Create cudnn-handle 0

conv 32 3 x 3/ 1 416 x 416 x 3 -> 416 x 416 x 32 0.299 BF
1 conv 64 3 x 3/ 2 416 x 416 x 32 -> 208 x 208 x 64 1.595 BF
2 conv 32 1 x 1/ 1 208 x 208 x 64 -> 208 x 208 x 32 0.177 BF
3 conv 64 3 x 3/ 1 208 x 208 x 32 -> 208 x 208 x 64 1.595 BF
4 Shortcut Layer: 1, wt = 0, wn = 0, outputs: 208 x 208 x 64 0.003 BF
5 conv 128 3 x 3/ 2 208 x 208 x 64 -> 104 x 104 x 128 1.595 BF
6 conv 64 1 x 1/ 1 104 x 104 x 128 -> 104 x 104 x 64 0.177 BF
7 conv 128 3 x 3/ 1 104 x 104 x 64 -> 104 x 104 x 128 1.595 BF
8 Shortcut Layer: 5, wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
9 conv 64 1 x 1/ 1 104 x 104 x 128 -> 104 x 104 x 64 0.177 BF
10 conv 128 3 x 3/ 1 104 x 104 x 64 -> 104 x 104 x 128 1.595 BF
11 Shortcut Layer: 8, wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
12 conv 256 3 x 3/ 2 104 x 104 x 128 -> 52 x 52 x 256 1.595 BF
13 conv 128 1 x 1/ 1 52 x 52 x 256 -> 52 x 52 x 128 0.177 BF
14 conv 256 3 x 3/ 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BF
15 Shortcut Layer: 12, wt = 0, wn = 0, outputs: 52 x 52 x 256 0.001 BF

95 route 91 -> 26 x 26 x 256
96 conv 128 1 x 1/ 1 26 x 26 x 256 -> 26 x 26 x 128 0.044 BF
97 upsample 2x 26 x 26 x 128 -> 52 x 52 x 128
98 route 97 36 -> 52 x 52 x 384
99 conv 128 1 x 1/ 1 52 x 52 x 384 -> 52 x 52 x 128 0.266 BF
100 conv 256 3 x 3/ 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BF
101 conv 128 1 x 1/ 1 52 x 52 x 256 -> 52 x 52 x 128 0.177 BF
102 conv 256 3 x 3/ 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BF
103 conv 128 1 x 1/ 1 52 x 52 x 256 -> 52 x 52 x 128 0.177 BF
104 conv 256 3 x 3/ 1 52 x 52 x 128 -> 52 x 52 x 256 1.595 BF
105 conv 255 1 x 1/ 1 52 x 52 x 256 -> 52 x 52 x 255 0.353 BF
106 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.00
Total BFLOPS 65.879
avg_outputs = 532444
Allocate additional workspace_size = 134.22 MB
Loading weights from yolov3.weights...
seen 64, trained: 32013 K-images (500 Kilo-batches_64)
Done! Loaded 107 layers from weights-file
Detection layer: 82 - type = 28
Detection layer: 94 - type = 28
Detection layer: 106 - type = 28
data/dog.jpg: Predicted in 8.227000 milli-seconds.
bicycle: 99%
dog: 100%
truck: 93%
(opencv_dnn_cuda) user ~ /Desktop/AlexeyAB master ±