DL Seminar

FaceNet

A Unified Embedding for Face Recognition and Clustering

한양대학교 HANYANG UNIVERSITY

인공지능 연구실
김지성

Index

1. Introduction
2. Method
3. Experiments
4. Demonstrate

Introduction

Face Recognition

1. 얼굴 찾기

2. 얼굴 정렬

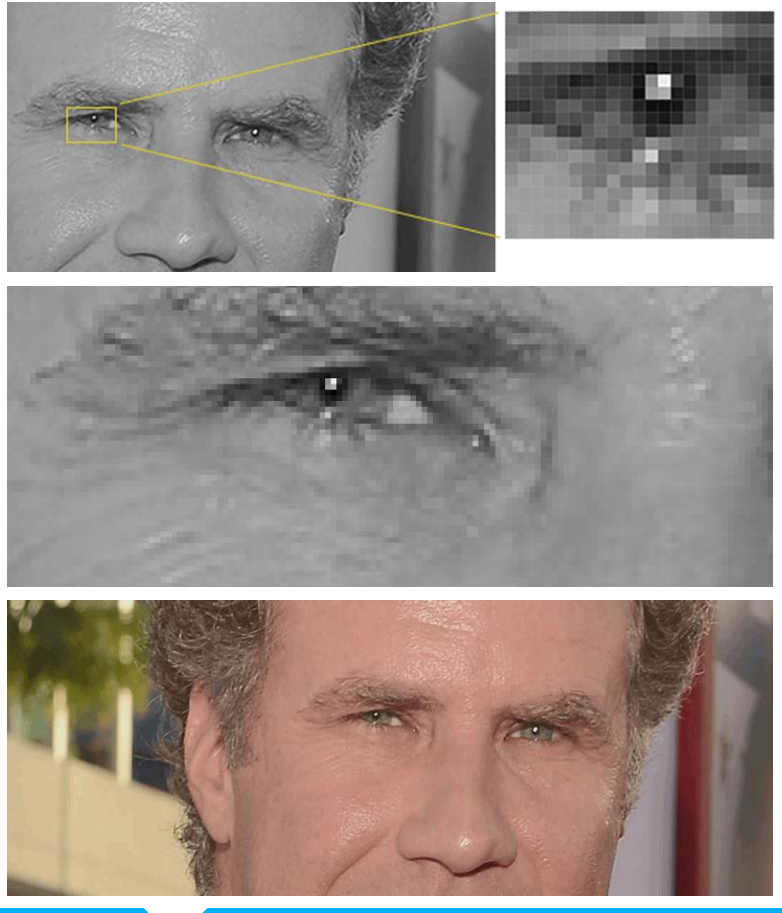
3. 얼굴 인코딩

4. 얼굴 비교

얼굴인식의 전통적인 방법

Introduction

HOG(Histogram of Oriented Gradients)



HOG를 이용한 패턴 매칭을 하면 물체의 밝기 변화에 보다 강력하게 경계선을 추출할 수 있습니다.

1. 색상정보가 필요 없으므로 이미지를 흑백으로 변환합니다.

2. 한 픽셀과 이 픽셀을 둘러싸고 있는(상하좌우) 픽셀들과 비교해서 더 어두워지는 쪽을 향하는 선을 그립니다.

3. 이미지의 모든 픽셀에 위 작업을 반복합니다.

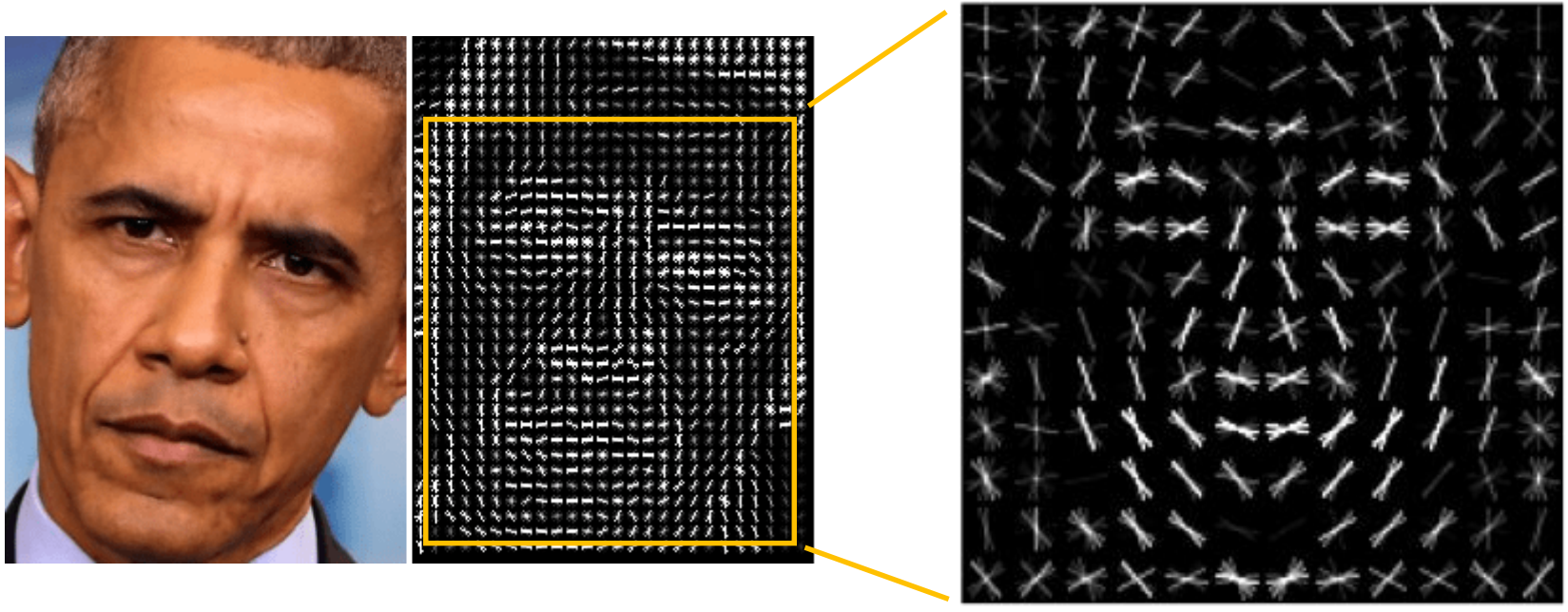
이미지의 모든 픽셀에 위 프로세스를 반복하면 모든 픽셀이 선으로 바뀌게 되고 이러한 선들을 Gradients라고 부릅니다.

그러나 위 정보는 너무 자세하기 때문에 패턴을 쉽게 알 수 있도록 더 High-Level에서 밝음->어둠의 흐름만을 보는 것이 더 좋습니다.

이를 위해 이미지를 16x16픽셀 의 정사각형들로 분해하고, 정사각형에서 얼마나 많은 Gradient가 주요 방향을 가리키고있는지 세어 그려줍니다.

Introduction

HOG(Histogram of Oriented Gradients)

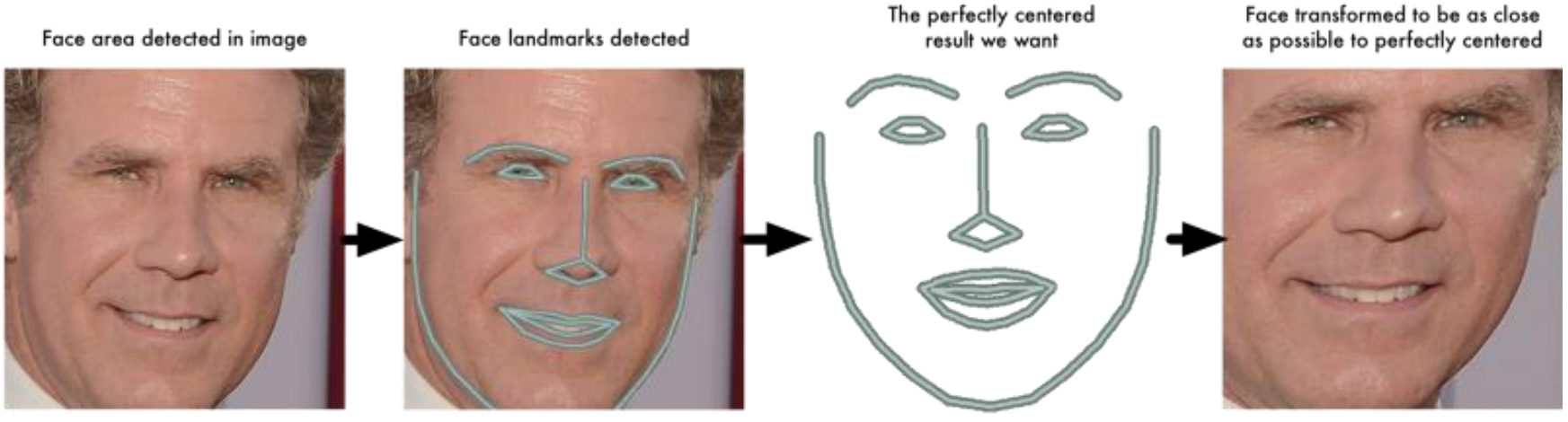


그런 다음, HOG표현된 이미지와 많은 얼굴을 학습한 HOG패턴을 비교하여 사진에서 얼굴이 어디에 존재하는지 파악할 수 있습니다.

Introduction

3D Warp

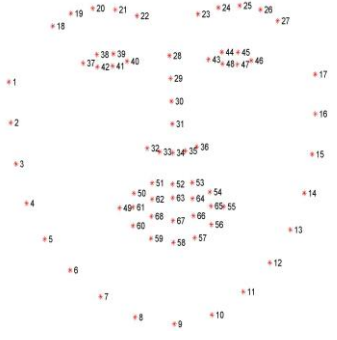
기울어진 얼굴을 정면을 바라보는 얼굴로 만드는 작업입니다. 따라서 이미 정면 얼굴이라면 필요 없는 작업입니다.

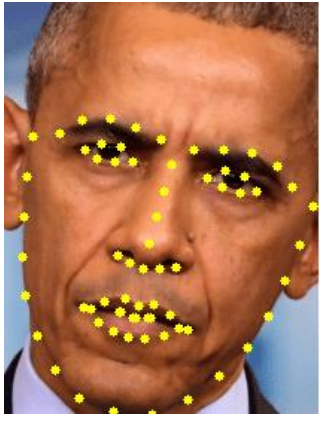


1. 이전 과정에서 추출한 얼굴영역을 준비합니다.
2. 얼굴 영역에서 얼굴의 요소를 의미하는 랜드마크를 찾아냅니다.
3. 기울어진 얼굴이 정면을 바라볼 수 있도록 이미지를 변형합니다. 3d warp

Introduction

3D Warp





Face Landmark Example

```
version: 1
n_points:  68
{
144.524547 171.476994
150.122948 246.290660
169.425571 325.523453
192.661317 394.572785
237.380705 459.609720
296.262893 501.774631
362.318259 530.076867
433.724968 544.129317
```

```
467.581566 390.331365
486.462048 387.782468
512.871913 389.980182
486.033520 416.889430
464.645172 424.310959
447.685618 424.427189
}
```

Helen Dataset Example

Introduction

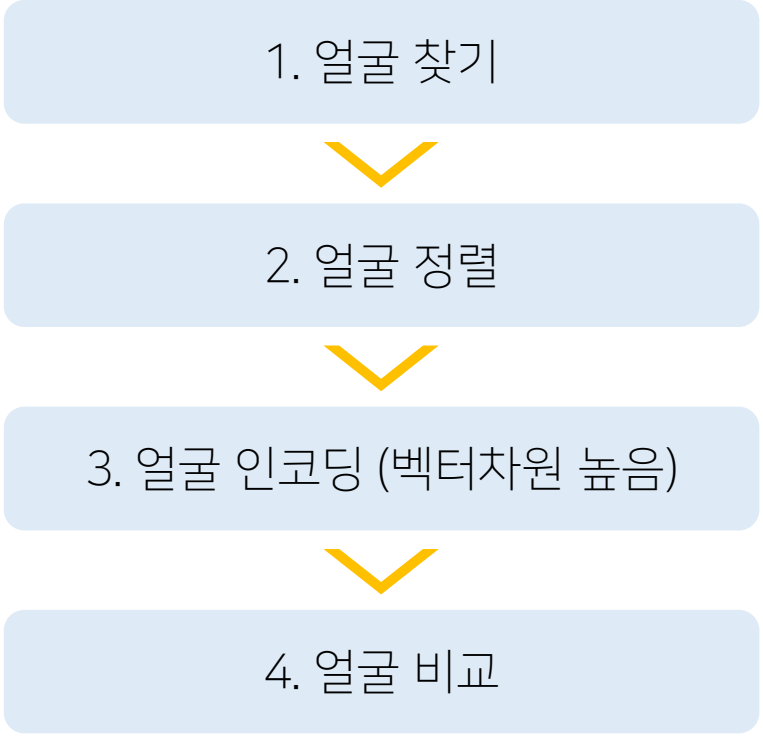
Embedding

array([-0.07440512, 0.13833548, 0.01550988, -0.04143589, -0.11708137,
0.01741652, -0.09219746, -0.0411015 , 0.12901564, -0.03510666,
0.26017255, -0.01175268, -0.23214489, -0.10993981, -0.06433399,
0.14533579, -0.18374404, -0.0706907 , 0.01960303, 0.03927957,
0.17917837, 0.10840175, 0.06728961, 0.01386871, -0.0921066 ,
-0.32157087, -0.06875613, -0.11010353, 0.02105946, -0.09806988,
-0.09223023, -0.01714757, -0.16107245, -0.04865564, 0.05062422,
0.04144309, -0.03346116, -0.03011878, 0.15263124, 0.01069992,
-0.23673587, 0.05740198, 0.03050802, 0.23846291, 0.20607698,
0.01160336, 0.00998583, -0.15661725, 0.09741502, -0.11773828,
0.08130169, 0.15210505, 0.14222656, 0.02321066, 0.00600557,
-0.07693202, -0.02959017, 0.15295519, -0.13042481, 0.03233573,
0.1088061 , -0.05199346, -0.01501178, -0.08011279, 0.17588341,
0.02202863, -0.12124902, -0.26303217, 0.06608438, -0.123976 ,
-0.14779539, 0.1516934 , -0.16154116, -0.1716671 , -0.25228465,
0.01856503, 0.36660314, 0.04856375, -0.18907131, 0.05604936,
-0.05154709, -0.04398936, 0.08519959, 0.14640823, 0.00993889,
0.02739849, -0.11102622, 0.01848426, 0.23328975, -0.11351117,
-0.04641433, 0.22538319, -0.00492372, 0.10828383, 0.02823116,
0.02502055, -0.02946196, 0.0702677 , -0.09549975, -0.0334047 ,
0.00996118, -0.08729131, -0.04567208, 0.09973253, -0.14260028,
0.10422572, -0.00379006, 0.05201333, 0.02785442, -0.05933321,
-0.09983464, -0.02418252, 0.13822252, -0.24259827, 0.2536535 ,
0.12104575, 0.14091188, 0.07011457, 0.10865125, 0.04752614,
0.02150964, -0.04581762, -0.23496597, -0.01055925, 0.11252803,
-0.05433004, 0.10194337, -0.02596316])

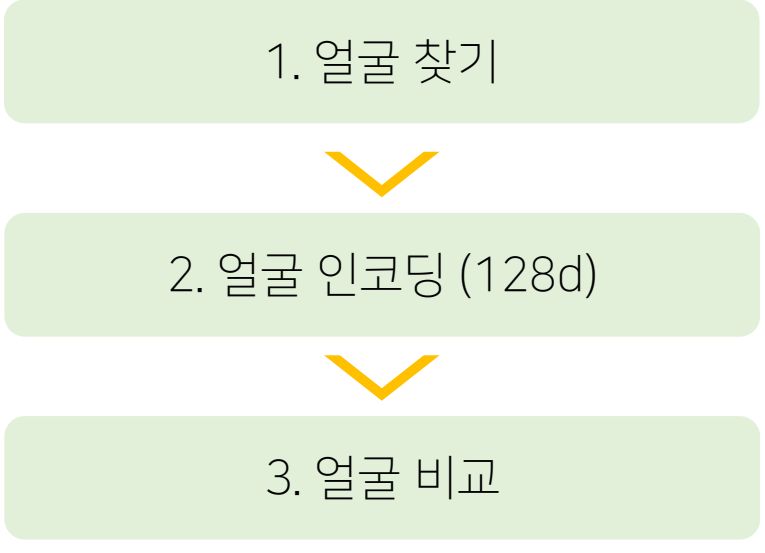
array([-0.08902847, 0.14762324, 0.05718813, -0.05012994, -0.09676401,
0.02028476, -0.08940892, -0.0386399 , 0.10385386, -0.031321 ,
0.2729257 , -0.02359607, -0.23816103, -0.09732635, -0.03333118,
0.1333721 , -0.19538365, -0.07668579, 0.00275577, 0.03931012,
0.15097558, 0.08741103, 0.0678171 , 0.01446525, -0.09825172,
-0.33090472, -0.06750867, -0.10570621, 0.01662679, -0.11775655,
-0.10916033, -0.04665562, -0.1787585 , -0.05166516, 0.04875493,
0.0426253 , -0.0591871 , -0.04663718, 0.16380328, 0.00791238,
-0.22012679, 0.04296945, 0.04917528, 0.23990902, 0.21356051,
0.00606929, 0.00836444, -0.13134141, 0.09845343, -0.162596 ,
0.11051578, 0.15520622, 0.12857951, 0.05687075, 0.01556353,
-0.09330104, -0.04209765, 0.14811578, -0.10172325, 0.06637652,
0.11120091, -0.05348029, -0.02427355, -0.06907345, 0.17457779,
0.04438576, -0.13614309, -0.27176958, 0.04076207, -0.10192671,
-0.13134097, 0.16820309, -0.17274556, -0.17039801, -0.25199199,
0.0466072 , 0.38125145, 0.03306871, -0.19173753, 0.01305585,
-0.07093169, -0.05639979, 0.07696941, 0.15101001, 0.02203412,
0.0188837 , -0.10816664, 0.04892536, 0.23276457, -0.10933174,
-0.06033619, 0.2151441 , -0.01290991, 0.10794014, 0.02028765,
0.05325544, -0.03414371, 0.055746 , -0.10889978, -0.02816046,
0.01456824, -0.07452301, -0.06755616, 0.11638117, -0.13528842,
0.10086828, 0.00977207, 0.03836292, 0.02850841, -0.05801099,
-0.10640397, -0.03940248, 0.12043379, -0.25706992, 0.24780291,
0.14584672, 0.13685006, 0.05027201, 0.09039105, 0.03249723,
0.04752009, -0.02310574, -0.21513188, -0.00798578, 0.09137717,
-0.01307906, 0.06952387, -0.03653527])

Introduction

FaceNet



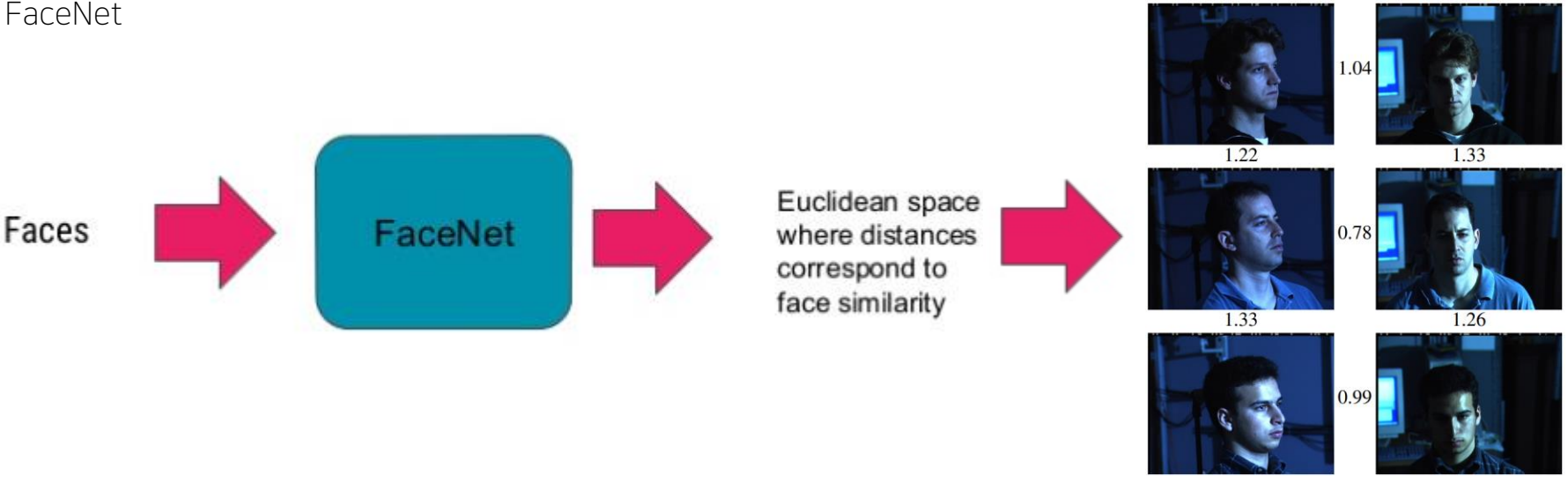
얼굴인식의 전통적인 방법



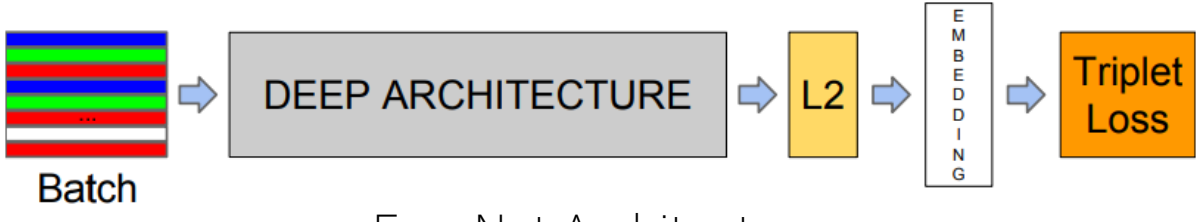
FaceNet

Introduction

FaceNet



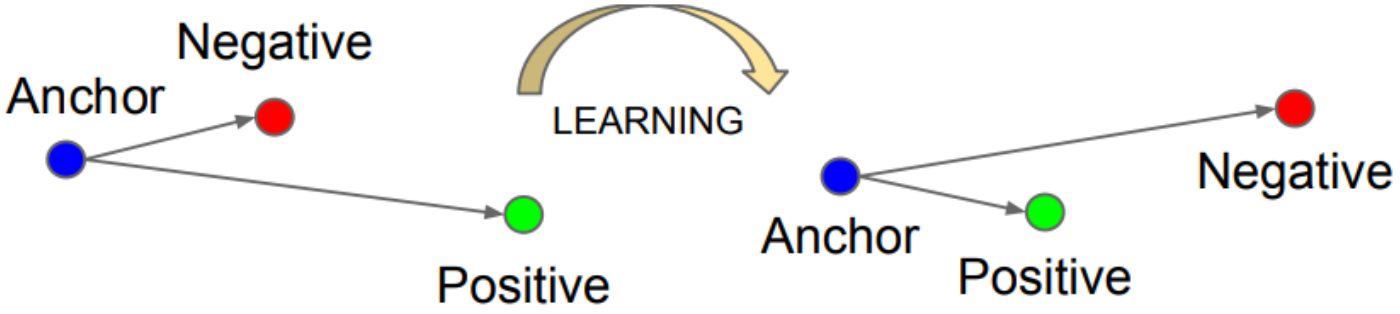
FaceNet BlackBox



FaceNet Architecture

Method

FaceNet



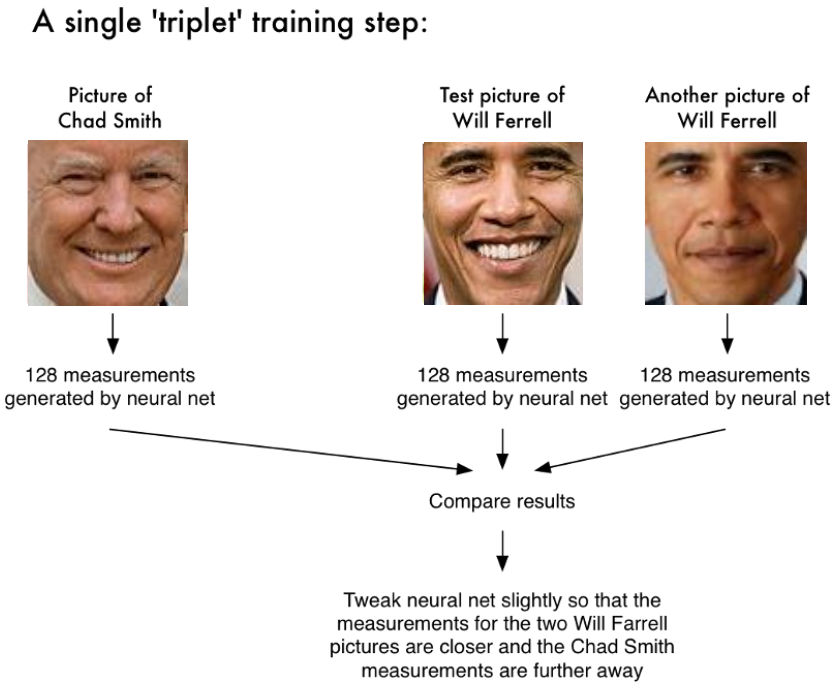
Triplet Loss : 세 개의 데이터

- Anchor(x_i^a) : 기준 얼굴
- Positive(x_i^p) : 기준과 같은 인물의 얼굴
- Negative(x_i^n) : 기준과 다른 인물의 얼굴

기준 얼굴과 같은 얼굴은 가깝도록, 기준 얼굴과 다른 얼굴은 멀도록

Method

Triplet



학습

Input

1. 알고있는 인물 A의 사진(Anchor)
2. 동일한 인물 A의 1과 다른 사진(Positive)
3. 전혀 다른 인물 B의 사진(Negative)

알고리즘은 세 개의 이미지 각각에 대해 현재 생성하고 있는 측정값을 확인합니다.

1과 2 이미지에 대해 생성한 측정값은 서로 가깝게,
2와 3의 측정 값은 서로 멀게 신경망을 조정합니다.

Output
128차원의 벡터(추천)

Method

Triplet - Loss

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 \,, \tag{1}$$

- x : 이미지
- $f(x) \in \mathbb{R}^d$: 임베딩메서드
- α : 마진

- Anchor(x_i^a) : 기준 얼굴
- Positive(x_i^p) : 기준과 같은 인물의 얼굴
- Negative(x_i^n) : 기준과 다른 인물의 얼굴

Anchor와 Negative의 거리가 Anchor와 Positive의 거리보다 α 만큼 떨어져 있고 싶다!

$$\forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in \mathcal{T} \,. \tag{2}$$

식1을 만족하는 임베딩된 세쌍은 트리플릿으로 묶여 트레이닝셋T의 원소가 됨

$$\text{LOSS} = \sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+ \,. \tag{3}$$

Method

Triplet – Triplet Selection

두 명의 사람이 각 10장의 사진을 가질 때 가능한 Triplet의 수 :
Class x Anchor x Positive x Negative = 2 x 10 x 9 x 10 = 1800

대량의 학습 데이터에서 너무 많은 Triplet이 전부 학습에 도움을 주지는 않음
-> 수렴이 느려짐
따라서 어려운 문제를 골라서 해결하기 위해 Hard Negative, Hard Positive를 선택해야 함

x_i^a Anchor가 주어졌을 때

$\text{argmin}_{x_i^n} \|f(x_i^a) - f(x_i^n)\|_2^2$ Hard Negative : 가장 가까이 있는 Negative

$\text{argmax}_{x_i^p} \|f(x_i^a) - f(x_i^p)\|_2^2$ Hard Positive : 가장 멀리 있는 Positive

Method

Triplet – Triplet Selection(Detail)

- Mini-batch에 약 40개의 identity
- Hard-positive 는 사용하지 않음
  - 모든 positive 사용
  - 실험해보니 더 안정적인 결과가 나옴
  - 초반에 더 빠르게 수렴함
- 학습초반에 가장 Hard한 negative를 선택하면 local-minima에 빠지기 쉽다
  - 학습 초반에 너무 어려운 문제를 해결하기 힘들다
  - Semi-hard negative를 선택해야함
    - Anchor와 positive를 정한 뒤 positive보다 먼 negative 선택
    - Positive와 거리가 비슷해서 Hard라고 말할 수 있다

$$\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 \ . \qquad (4)$$

Experiments

Experiments Define

Train dataset :
- 서술되지 않음

Test Dataset :
- Hold-out test set
- Personal photos

Evaluation
- VAL(d) : Validation Rate
  같은 사람을 같다고 예측한 비율
- FAR(d) : False Accept Rate
  다른 사람을 같다고 예측한 비율

Experiments

CNN Model

실험에 사용된 CNN 모델

architecture	VAL
NN1 (Zeiler&Fergus 220×220)	87.9% ± 1.9
NN2 (Inception 224×224)	89.4% ± 1.6
NN3 (Inception 160×160)	88.3% ± 1.7
NN4 (Inception 96×96)	82.0% ± 2.3
NNS1 (mini Inception 165×165)	82.4% ± 2.4
NNS2 (tiny Inception 140×116)	51.9% ± 2.9

Category1 : Zeiler & Fergus : ZF Net 기반 모델

- NN1
  - 1x1 convolution

Category 2 : GoogLeNet 기반 모델

- NN2
  - NN1과 비교
    - 약 20배 적은 파라미터
    - 최대 5배 까지 적은 FLOPS
  - NN2~4
    - 입력 크기 220x220, 160x160, 96x96
  - NNS1~4
    - 모바일을 위한 경량 모델

Experiments

CNN Model

실험에 사용된 CNN 모델

layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

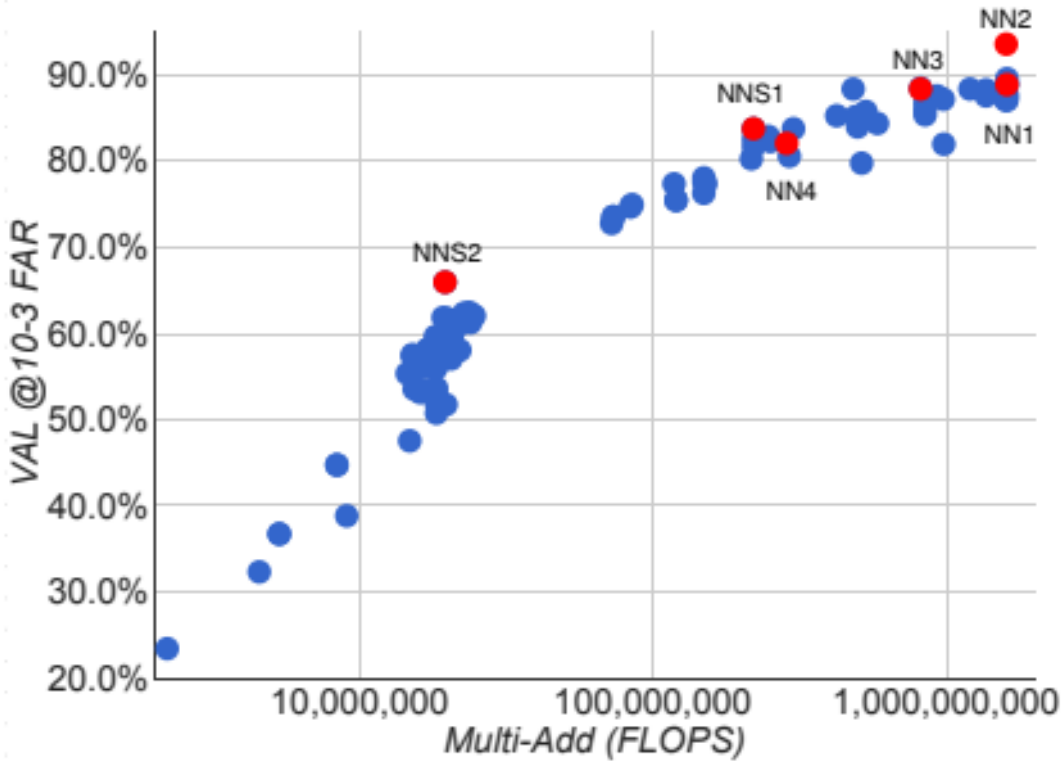
Zeiler & Fergus Model : ZF Net
-> AlexNet의 개선버전

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

GoogLeNet
-> Inception Module 개념 도입

Experiments

FLOPS vs. Accuracy



FLOPS vs. Accuracy

Experiments

FLOPS vs. Accuracy

jpeg q	val-rate
10	67.3%
20	81.4%
30	83.9%
50	85.5%
70	86.1%
90	86.5%

이미지 품질에 따른 정확도

#pixels	val-rate
1,600	37.8%
6,400	79.5%
14,400	84.5%
25,600	85.7%
65,536	86.4%

#dims	VAL
64	86.8% ± 1.7
128	87.9% ± 1.9
256	87.7% ± 1.9
512	85.6% ± 2.0

임베딩 벡터 차원에 따른 정확도

#training images	VAL
2,600,000	76.3%
26,000,000	85.1%
52,000,000	85.1%
260,000,000	86.2%

데이터셋 양에 따른 정확도

시 연


감사합니다.