인공지능론 Term-Project 최종 보고서

2017100872 김수빈

2017100878 김익환

2017100892 박근태

2017100920 장현욱

목 차

현재 기술 수준 및 기술 동향

CNN를 활용한 얼굴 인식 기술

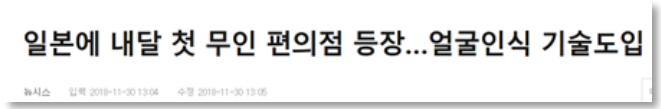
딥러닝 기법 및 모델 설계

Keras를 활용한 모델 설계 및 결과

미래 개선 방안 제시

모델 개선 방안 및 CNN의 미래 개선 방안

현재 기술 수준 및 기술 동향


일본에 내달 첫 무인 편의점 등장...얼굴인식 기술도입

뉴시스 입력 2018-11-30 13:04 수정 2018-11-30 13:05

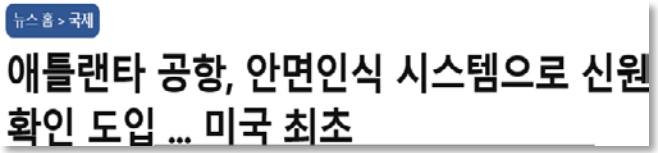

얼굴인식 기술, 중국인의 일상에 '성큼'

좋아요 11개 | 입력 : 2018-03-28 13:44

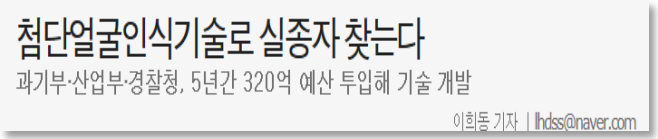
#시큐리티헬드 #생체인식 #바이오인식 #바이오메트릭스 #출입통제 #지문인식 #얼굴인식

중국 생체인식 시장동향 살펴보니

[보안뉴스 김성미 기자] 애플이 지난해 9월 3D 얼굴인식 잠금해제기능(Face ID)을 보유한 신제품 '아이폰 X'를 출시하면서 중국에서 얼굴인식 기술 응용이 산업계의 화두로 떠올랐다. 사오미는 애플의 신제품 발표회 하루 전날에 중국 스마트폰 제조사 중 처음으로 얼굴인식기술을 탑재한 '미노트(Mi note) 3'를 선보이기도 했다. 여기에서는 중국 얼굴인식 시장의 현재를 살펴본다.

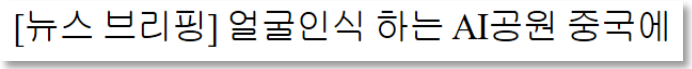

뉴스 홈 > 국제

애틀랜타 공항, 안면인식 시스템으로 신원 확인 도입 ... 미국 최초


첨단얼굴인식기술로 실종자 찾는다

과기부·산업부·경찰청, 5년간 320억 예산 투입해 기술 개발

이희동 기자 | lhdss@naver.com

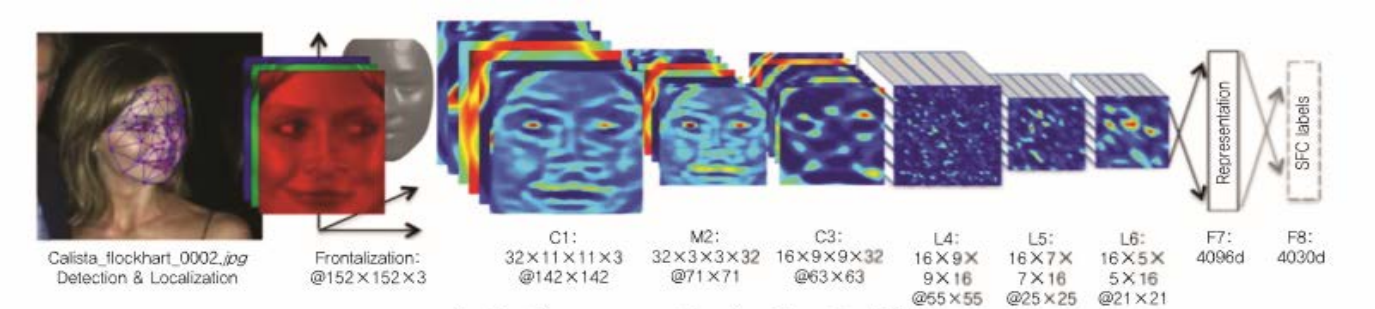

[뉴스 브리핑] 얼굴인식 하는 AI공원 중국에

얼굴인식 기술

얼굴을 포함하는 입력 정지 영상 또는 비디오에 대해 얼굴 영역의 자동적인 검출 및 분석을 통해 **해당 얼굴이 어떤 인물인지 판별해내는 분야**이다.

많은 수의 층으로 구성된 깊은 신경망 구조를 대용량의 데이터를 이용하여 학습시키는 기술인 딥러닝 기술의 발달에 힘입어 영상인식 기술이 매우 빠른 속도로 발전하고 있고, 여러 분야에 적용 중에 있다.

현재 기술 수준 및 기술 동향



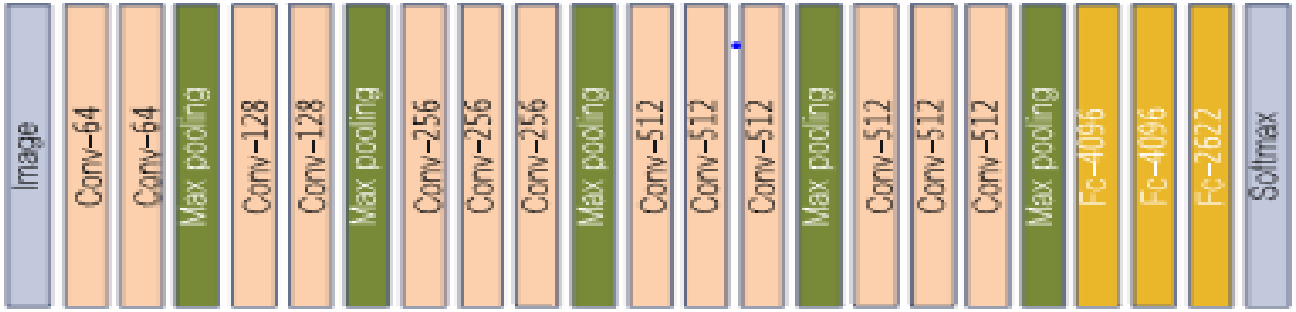
(그림 3) Deepface 딥 네트워크 구조[1]

Facebook DeepFace

딥러닝 기술이 얼굴인식에서 처음으로 접목된 연구는 2014년 CVPR에서 발표된 Facebook의 DeepFace 이다. **현재 페이스북에 인물 사진을 올리면 얼굴 부분에 '친구를 태그하시겠습니까?'라고 메시지가 뜨는데 이 것이 바로 DeepFace를 활용한 예이다.**

DeepFace에서는 사전에 학습된 3D 얼굴 기하 모델을 이용하여 랜드마크 추출 후에 Affine 변환에 의해 얼굴을 편집한다. 그 후 Convolution 신경망으로 데이터를 학습하게 된다. 이는 매우 많은 수의 신경망 학습이 필요로 한다.

현재 기술 수준 및 기술 동향

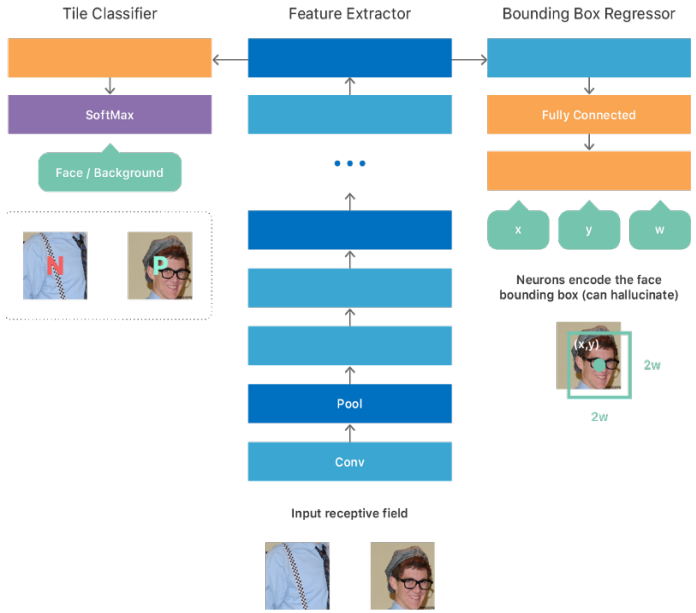


(그림 4) VGGFace 딥 네트워크 구조

DeepFace 이후 최근에 등장한 구조는 옥스포드 대학에서 제안한 딥 네트워크 구조이다. 여러 개의 컨볼루션 필터를 이용하여 학습시킴으로써 대용량 사진 데이터셋을 학습하고 영상 분석을 실행한다.

기존의 DeepFace분석보다 향상된 얼굴 인식이 수행되어 질 수 있다.

현재 기술 수준 및 기술 동향



Apple Face ID DeepFace

아이폰X에서 처음 탑재된 애플의 보안 기술 시스템으로 사용자의 얼굴을 인식해 보안을 해제하는 안면 인증 기술이다. 이는 뉴욕 대학의 연구진들이 개발한 모델인 OverFeat을 기반으로 한다.

이미지의 객체를 분류, 얼굴의 위치를 지정하고 감지를 하는 DCN (Deep Convolution Neural Network) 기술을 활용했는데, **이를 통해 사용자가 모자를 쓰거나, 안경을 쓰는 등의 외모 변화를 기계 스스로 학습해 적응할 수 있다.**

숨은 현욱 얼굴 찾기

딥러닝 기법 및 모델 설계

데이터 전처리

얼굴 부분으로 한정해서
사진들을 자른다.

학습 모델 만들기

현욱이와 다른 사람의 사진을
데이터로 주어 모델을 학습시킨다.

학습 모델 평가하기

Test Data로 학습 시킨 모델이
얼마만큼 정확한지를 파악한다.

예측 모델 만들기

새로운 사진을 넣어, 이 사진이
현욱이인지 아닌지를 구별한다.

01
데이터 전처리

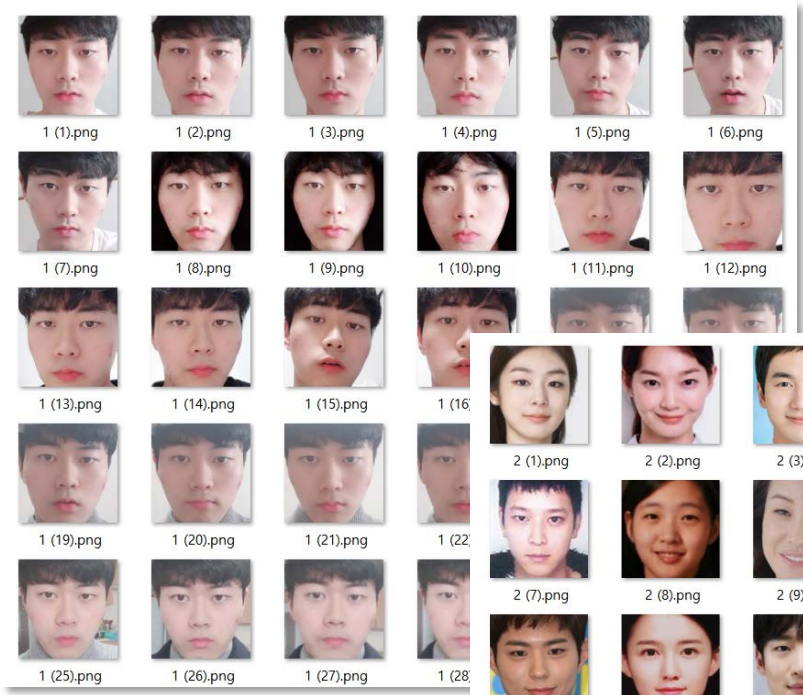
1. 가공되지 않은 얼굴 사진(현욱 + 다른 사람)을 수집한다.

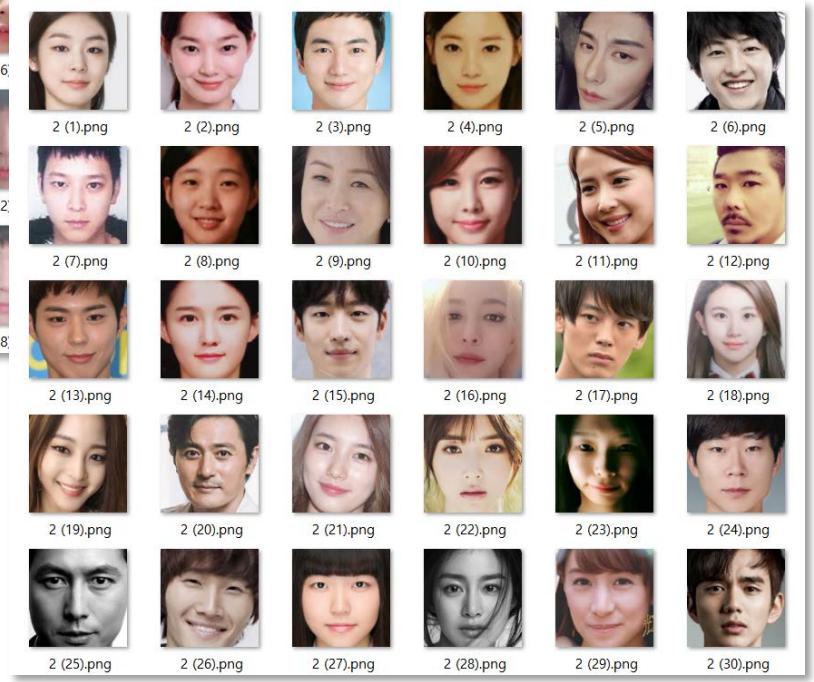
2. 분류의 정확도를 높이기 위해 OpenCV를 활용해서, 얼굴만 나오게 사진을 자른다.

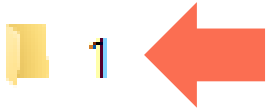
```
import cv2
import sys

num=0
imgNum  = 0
while num<1:
    num+=1
    print(num)

    image_file = "1 ("+str(num)+").jpg"
    cascade_file = "haarcascade_frontalface_alt.xml"


    image = cv2.imread(image_file)
    image_gs = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    cascade = cv2.CascadeClassifier(cascade_file)
    face_list = cascade.detectMultiScale(image_gs, scaleFactor = 1.1, minNeighbors = 1, minSize = (150, 150))

    if len(face_list) > 0:
        print(face_list)
        color = (0, 0, 255)
        for (x,y,w,h) in face_list:
            cropped = image[y:y + h, x:x + w]


        # 이미지를 저장
            cv2.imwrite("thumbnail" + str(imgNum) + ".png", cropped)
            imgNum += 1

    else:
        print("no face")
```

01
데이터 전처리

1. 현욱이의 얼굴 사진을 폴더 0으로, 다른 사람들의 얼굴사진을 폴더 1로 분류해 데이터를 저장한다.







02
학습모델 만들기

1. Keras를 불러온다
2. 64 X 64 사이즈의 input data를 넣도록 모델을 만든다.
3. Size가 3 X 3인 Kernel을 32개 사용하고, 활성함수는 ReLU function을 사용해 Convolution한다.
4. Kernel Size 2 X 2로 MaxPooling한다.
5. Size가 3 X 3인 Kernel을 64개 사용하고, 활성함수는 ReLU function을 사용해 2번째 Convolution한다.
6. Kernel Size 2 X 2로 2번째 MaxPooling한다.
7. 과적합을 방지하기 위해 Dropout을 하고 Flatten한다.

```
import keras
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Dense

model = Sequential()
#model 만들기


model.add(Conv2D(32, kernel_size=(3, 3), input_shape = (64, 64, 3), activation = 'relu'))
# Convolution                                          RGB로 진행하므로 3개의 층

model.add(MaxPooling2D(pool_size = (2, 2)))
# MaxPooling

model.add(Conv2D(64, kernel_size=(3, 3), activation = 'relu'))
# 2번째 Convolution

model.add(MaxPooling2D(pool_size = (2, 2)))
# 2번째 MaxPooling

model.add(Dropout(0.1))
#과적합 방지용 Dropout

model.add(Flatten())
# Flatten
```

02
학습모델 만들기

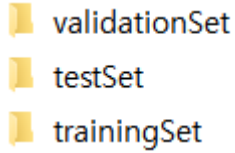
1. Flatten된 데이터를 128개의 Hidden Unit과 Fully Connect한다.
2. 과적합을 방지하기 위한 Dropout을 실행한다.
3. Output 결과는 Binary이므로 활성함수로 Sigmoid를 사용한다.
4. 모델을 Optimizer를 Adam, Loss를 Binary Entropy를 사용하고, accuracy를 측정한다.
5. Patience가 3인 Callback을 실행한다.
6. ImageGenerator를 RGB scale를 0과 255에서 0과 1 사이로 변환한다.

```
model.add(Dense(units = 128, activation = 'relu'))
#Hidden units

model.add(Dropout(0.1))
#과적합 방지용 Dropout

model.add(Dense(units = 1, activation = 'sigmoid'))
# Outputs


model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
# Compiling

from keras.callbacks import EarlyStopping
early_stop=EarlyStopping(monitor='val_loss',patience=3, mode='min')
#patience설정

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255)
#rescale=RGB크기를 0-255를 0-1사이로 변환

validation_datagen = ImageDataGenerator(rescale = 1./255)
#rescale=RGB크기를 0-255를 0-1사이로 변환

test_datagen = ImageDataGenerator(rescale = 1./255)
#rescale=RGB크기를 0-255를 0-1사이로 변환
```

02
학습모델 만들기

1. Training Set, Validation Set, Test Set 안의 데이터를 C드라이브에서 불러온다.
2. 세 폴더 안에 든 사진들을 64 X 64 사이즈로 변환하고, Batch Size를 2, 그리고 Binary Class로 설정한다.
3. 모델을 Training Data와 Validation Data를 활용해 epoch는 10, callback를 사용해 학습시킨다.

```
training_set = train_datagen.flow_from_directory('C:/trainingSet/',
                                                 target_size = (64, 64),
                                                 batch_size = 2,
                                                 class_mode = 'binary')
#C드라이브의 trainingSet에서 이미지를 불러 이미지 사이즈를 64X64로 크기 조정, batch size는 2, class는 binary

validation_set = validation_datagen.flow_from_directory('C:/validationSet/',
                                                 target_size = (64, 64),
                                                 batch_size = 2,
                                                 class_mode = 'binary')
#C드라이브의 validationSet에서 이미지를 불러 이미지 사이즈를 64X64로 크기 조정, batch size는 2, class는 binary

test_set = test_datagen.flow_from_directory('C:/testSet/',
                                            target_size = (64, 64),
                                            batch_size = 2,
                                            class_mode = 'binary')
#C드라이브의 testSet에서 이미지를 불러 이미지 사이즈를 64X64로 크기 조정, batch size는 2, class는 binary

model.fit_generator(training_set,
                    validation_data=validation_set,
                    epochs = 10,
                    callbacks=[early_stop])
#training_set과 validation_set으로 model훈련, epoch는 10, callback사용
```



03
학습모델 평가하기

1. Test Data를 사용해서 학습시킨 모델의 Loss값과 Accuracy 값을 구하여 평가한다.

2. 전체 데이터 중 Training Data를 80%, Test Data를 20%로 나누고 Training Data 중 Validation Data를 20%로 나누었다.

3. Training Data 290개, Validation Data 68개, Test Data 90개를 (0, 1) 클래스로 나눈다.

4. Training Data의 Accuracy 97.93%, Validation Data의 Accuracy 97.06%, Test Data의 Accuracy 98.88%로 결과가 나왔다.

5. 이 모델을 저장한다.

```
score=model.evaluate_generator(test_set)
print('LOSS: ',score[0])
print('ACCURACY: ',score[1])

model.save('인공지능론 과제.h5')
```

```
Found 290 images belonging to 2 classes.
Found 68 images belonging to 2 classes.
Found 90 images belonging to 2 classes.
Epoch 1/10
145/145 [==============================] - 19s 130ms/step - loss: 0.7232 - acc: 0.6690 - val_loss: 0.2071 - val_acc: 0.9118
Epoch 2/10
145/145 [==============================] - 17s 116ms/step - loss: 0.2333 - acc: 0.9069 - val_loss: 0.1180 - val_acc: 0.9706
Epoch 3/10
145/145 [==============================] - 16s 113ms/step - loss: 0.1350 - acc: 0.9586 - val_loss: 0.0704 - val_acc: 0.9853
Epoch 4/10
145/145 [==============================] - 17s 114ms/step - loss: 0.0830 - acc: 0.9724 - val_loss: 0.0206 - val_acc: 1.0000
Epoch 5/10
145/145 [==============================] - 18s 127ms/step - loss: 0.0996 - acc: 0.9655 - val_loss: 0.1016 - val_acc: 0.9559
Epoch 6/10
145/145 [==============================] - 16s 113ms/step - loss: 0.0752 - acc: 0.9690 - val_loss: 0.0732 - val_acc: 0.9559
Epoch 7/10
145/145 [==============================] - 16s 111ms/step - loss: 0.0470 - acc: 0.9793 - val_loss: 0.0592 - val_acc: 0.9706
LOSS:  0.043182092464288266
ACCURACY:  0.9888888888888889
```

04
예측모델 만들기

1. 전 과정에서 저장한 학습 모델을 불러온다.
2. 이전에 없던 새로운 사진 데이터를 C드라이브에 "test.jpg"로 준다.
3. 사진 데이터를 OpenCV를 사용해 얼굴 부분만 잘라 "face image.png"로 저장한다.
4. "face image.png"를 OpenCV에서 읽고 사이즈를 64 X 64로 만든다.
5. 그 파일을 예측하도록 predict_classes 함수에 넣고 결과값을 print한다.

```
from keras.models import load_model
import cv2
import numpy as np


model = load_model('인공지능론 과제.h5')

image_file = "C:/test.jpg"
cascade_file = "haarcascade_frontalface_alt.xml"


image = cv2.imread(image_file)


cascade = cv2.CascadeClassifier(cascade_file)
face_list = cascade.detectMultiScale(image,scaleFactor = 1.1, minNeighbors = 1, minSize = (150, 150))

if len(face_list) > 0:
    print("face exist")
    for (x,y,w,h) in face_list:
        cropped = image[y:y + h, x:x + w]
        cv2.imwrite("face image.png", cropped)


else:
    print("no face")

img = cv2.imread('face image.png')
img = cv2.resize(img,(64,64))
img = np.reshape(img,[1,64,64,3])

classes = model.predict_classes(img)

print (classes)
```

05
결과

face exist
[[0]]

face exist
[[1]]

미래 개선 방안 제시

사진 1 : 정제되고 완벽한 상황의 데이터

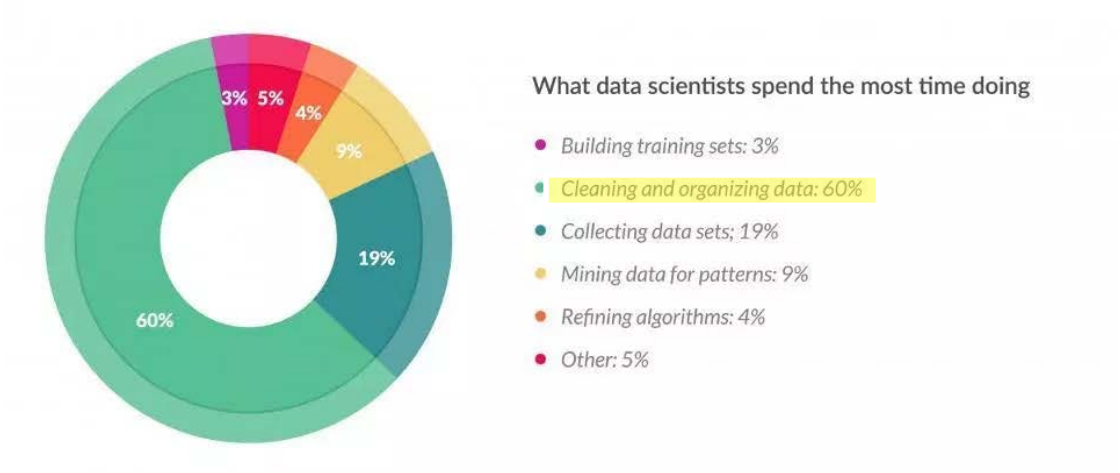
사진 2 : 날 것의 (노이즈가 많은) 상황의 데이터

학습 데이터 확보의 한계 및 극복

인공지능 구현에 필요한 데이터를 확보하는 것은 매우 중요하나 아주 어려운 과정이다. 많은 양을 확보하는 것도 중요하지만, 얼마나 질 좋은 데이터를 얻는지도 중요하기 때문이다.

질 좋은 데이터의 의미는 크게 두 가지인데 가장 먼저 다양성이다. 정제되고 완벽한 상황의 데이터 뿐만 아니라, **날 것의 데이터들이 같이 포함되어야, 다양한 상황에 대응할 수 있게 되고, 인공지능의 완성도를 높일 수 있다.**

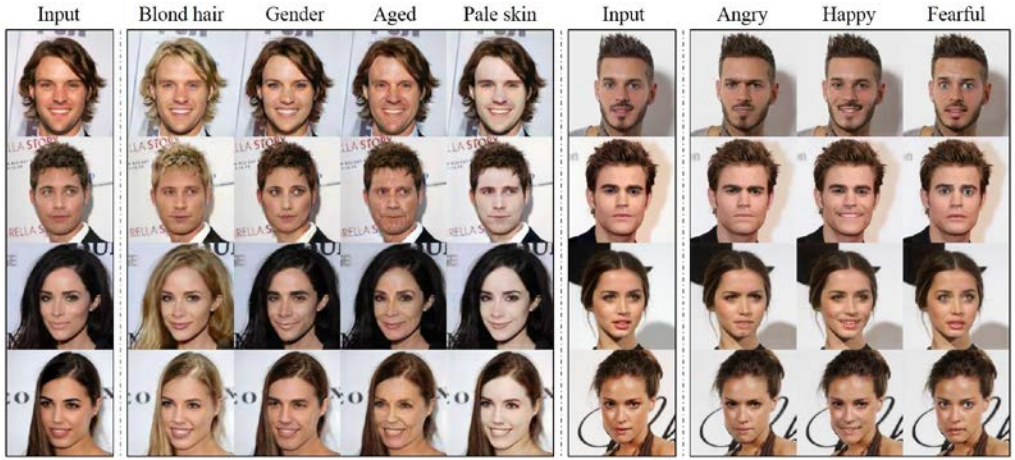
미래 개선 방안 제시



두 번째는 기계 학습에 가능한 형태로 준비되어야 한다는 점이다. 기계가 이해하기 위해서는 단순한 데이터도 전처리 과정이 필수적인데, 이를 수행하는데 드는 비용이 어마어마하다.

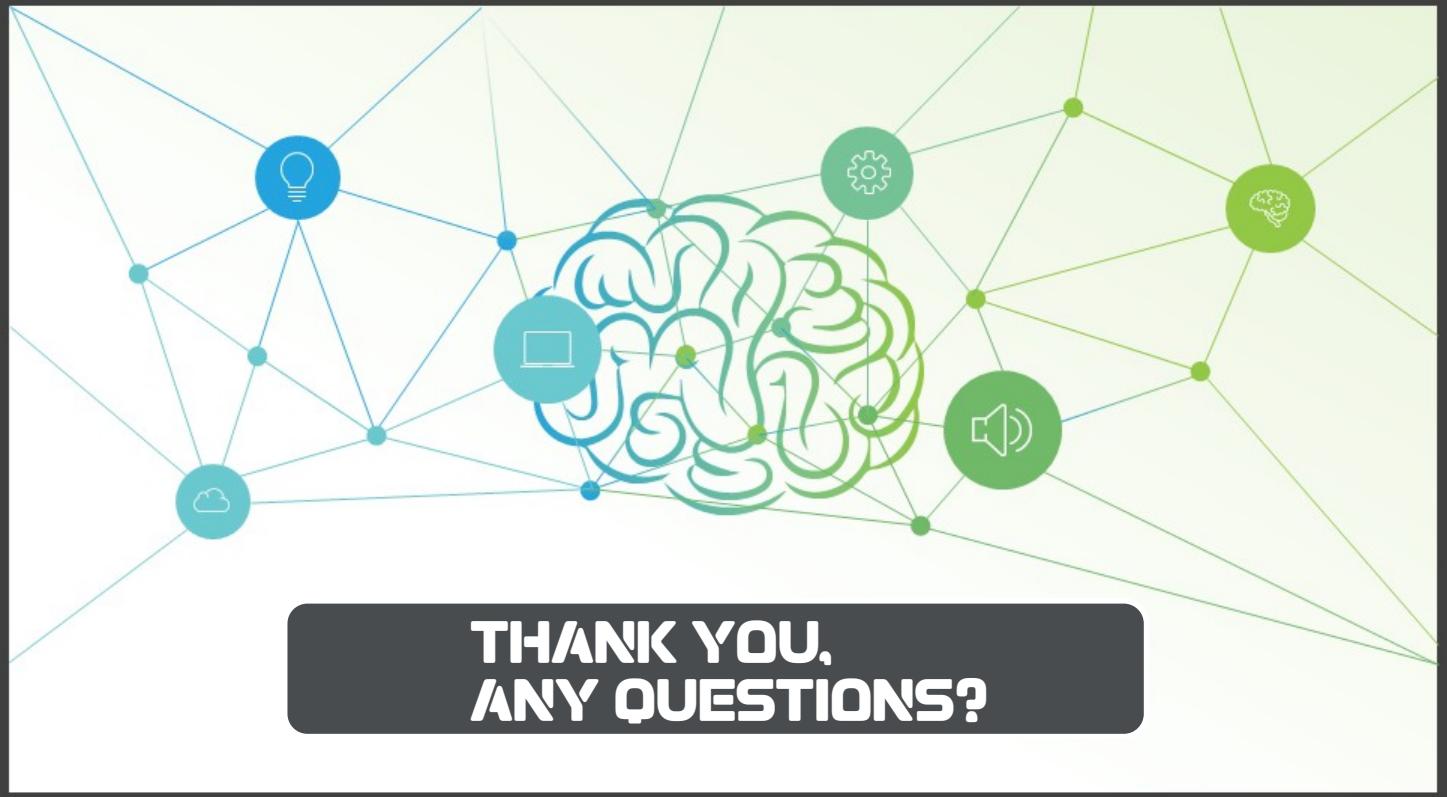
현재 인공 지능을 선도하는 거대 기업들은 데이터 확보를 위해 막대한 자금을 투자하고 있다. 그 예로 IBM이 의학 전문 자료를 확보하기 위해 의료 기관에 4조원 이상을 투자한 경우가 있다.

미래 개선 방안 제시



이를 극복하는 방법 중 하나는 **GAN 연구**로,
세상에 **존재하지 않는 가상의 데이터를 생성하는 인공지능이다.**

대상의 종류를 입력하면 해당하는 데이터를 자유롭게 생성한다. 이는 데이터 전처리 과정에 소요되는 시간과 비용을 획기적으로 줄일 수 있다.


THANK YOU,
ANY QUESTIONS?