2025 Fall – Machine Learning

트란항응아 (2022320166)

어용에르뎅 아미르랑 (2021320157)

# Final Report

# Personal Face Detection and Recognition System

## 1. 문제 정의

최근 얼굴 인식 기술은 보안 인증, 출입 통제 시스템, 개인화 서비스 등 다양한 분야에서 핵심적인 역할을 수행하고 있다. 기존의 얼굴 인식 시스템은 대규모 공개 데이터셋을 활용한 연구가 주를 이루었으나, 실제 환경에서는 개인 맞춤형(face personalization) 시스템의 필요성이 증가하고 있다.

본 프로젝트의 목표는 **단일 사용자 중심의 얼굴 검출(Face Detection) 및 얼굴 인식(Face Recognition) 시스템을 구현하는 것**이다. 이를 위해 다음과 같은 문제를 해결하고자 한다.

1. 이미지 내에서 얼굴이 존재하는지를 판별하고, 얼굴의 위치를 정확한 bounding box 형태로 예측한다.
2. 검출된 얼굴이 사전에 학습된 사용자 얼굴인지 여부를 이진 분류(binary classification) 방식으로 판단한다.

이 문제는 단순한 분류 문제를 넘어, **얼굴 존재 여부 분류와 얼굴 위치 회귀(regression)를 동시에 수행하는 멀티태스크 학습(multi-task learning)** 문제로 정의할 수 있다. 또한 소규모 데이터 환경에서 안정적인 성능을 확보하는 것이 주요 도전 과제이다.

## 2. 적용한 방법론 설명

### 2.1 데이터 수집 및 라벨링

본 프로젝트에서는 웹캠(OpenCV + Google Colab JavaScript API)을 활용하여 **직접 얼굴 이미지를 수집**하였다. 총 30 장의 원본 이미지를 촬영하였으며, 실제 사용 환경을 반영하기 위해 다양한 표정과 각도를 포함하였다.

라벨링 과정에서는 수작업 어노테이션 대신 **MTCNN 기반 pseudo-labeling 기법**을 적용하였다. MTCNN 모델을 사용하여 얼굴 bounding box 를 자동으로 검출하고, 해당

결과를 LabelMe 형식의 JSON 파일로 저장함으로써 효율적인 데이터 구축이 가능하도록 하였다.

```
⋯  Requesting camera... allow permission in the browser when prompted.
   Received 30 images; saving to data/images ...
   Done. Saved files:
    - data/images/93061260-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9352e59a-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/939fc04a-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/93ecc516-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9439c53c-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/94869b14-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/94d376c8-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/952054a2-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/956d273c-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/95b9fed6-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9606d670-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9653b9b8-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/96a092b0-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/96ed6ac2-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/973a4036-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/978718d4-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/97d3e678-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9820b98a-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/986d8a30-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/98ba5eaa-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/99073284-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/99540afa-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/99a0dfec-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/99edc0d2-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9a3abe6e-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9a87c9ca-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9ad4e002-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9b22017a-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9b6ee17a-da68-11f0-aa05-0242ac1c000c.jpg
    - data/images/9bbbb900-da68-11f0-aa05-0242ac1c000c.jpg
```

Figure 1. Image Collection

```
⋯  Found 30 images in data/images. Running detection on cuda ...
    [OK] 9b6ee17a-da68-11f0-aa05-0242ac1c000c.jpg -> 9b6ee17a-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 9bbbb900-da68-11f0-aa05-0242ac1c000c.jpg -> 9bbbb900-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 9b22017a-da68-11f0-aa05-0242ac1c000c.jpg -> 9b22017a-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 96ed6ac2-da68-11f0-aa05-0242ac1c000c.jpg -> 96ed6ac2-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 96a092b0-da68-11f0-aa05-0242ac1c000c.jpg -> 96a092b0-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 973a4036-da68-11f0-aa05-0242ac1c000c.jpg -> 973a4036-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 978718d4-da68-11f0-aa05-0242ac1c000c.jpg -> 978718d4-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 9352e59a-da68-11f0-aa05-0242ac1c000c.jpg -> 9352e59a-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 9606d670-da68-11f0-aa05-0242ac1c000c.jpg -> 9606d670-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 99edc0d2-da68-11f0-aa05-0242ac1c000c.jpg -> 99edc0d2-da68-11f0-aa05-0242ac1c000c.json, faces=0
    [OK] 93ecc516-da68-11f0-aa05-0242ac1c000c.jpg -> 93ecc516-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 956d273c-da68-11f0-aa05-0242ac1c000c.jpg -> 956d273c-da68-11f0-aa05-0242ac1c000c.json, faces=0
    [OK] 94d376c8-da68-11f0-aa05-0242ac1c000c.jpg -> 94d376c8-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 97d3e678-da68-11f0-aa05-0242ac1c000c.jpg -> 97d3e678-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 93061260-da68-11f0-aa05-0242ac1c000c.jpg -> 93061260-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 9653b9b8-da68-11f0-aa05-0242ac1c000c.jpg -> 9653b9b8-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 9a87c9ca-da68-11f0-aa05-0242ac1c000c.jpg -> 9a87c9ca-da68-11f0-aa05-0242ac1c000c.json, faces=2
    [OK] 939fc04a-da68-11f0-aa05-0242ac1c000c.jpg -> 939fc04a-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 99a0dfec-da68-11f0-aa05-0242ac1c000c.jpg -> 99a0dfec-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 9820b98a-da68-11f0-aa05-0242ac1c000c.jpg -> 9820b98a-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 952054a2-da68-11f0-aa05-0242ac1c000c.jpg -> 952054a2-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 94869b14-da68-11f0-aa05-0242ac1c000c.jpg -> 94869b14-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 9a3abe6e-da68-11f0-aa05-0242ac1c000c.jpg -> 9a3abe6e-da68-11f0-aa05-0242ac1c000c.json, faces=0
    [OK] 99540afa-da68-11f0-aa05-0242ac1c000c.jpg -> 99540afa-da68-11f0-aa05-0242ac1c000c.json, faces=0
    [OK] 9ad4e002-da68-11f0-aa05-0242ac1c000c.jpg -> 9ad4e002-da68-11f0-aa05-0242ac1c000c.json, faces=2
    [OK] 9439c53c-da68-11f0-aa05-0242ac1c000c.jpg -> 9439c53c-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 98ba5eaa-da68-11f0-aa05-0242ac1c000c.jpg -> 98ba5eaa-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 986d8a30-da68-11f0-aa05-0242ac1c000c.jpg -> 986d8a30-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 95b9fed6-da68-11f0-aa05-0242ac1c000c.jpg -> 95b9fed6-da68-11f0-aa05-0242ac1c000c.json, faces=1
    [OK] 99073284-da68-11f0-aa05-0242ac1c000c.jpg -> 99073284-da68-11f0-aa05-0242ac1c000c.json, faces=1
```

Figure 2. Autodetect bounding box data

또한 JSON 에는 얼굴 존재 여부를 나타내는 **클래스 라벨**이 포함된다.

- 1 → 얼굴 존재 (사용자 본인)
- 0 → 얼굴 없음 혹은 타인

예시:

```
{
  "filename": "image_23.jpg",
  "bbox": [120, 60, 340, 280],
  "class": 1
}
```

### 2.2 데이터 전처리 및 증강

수집된 데이터의 양이 제한적이기 때문에, **Albumentations 라이브러리**를 이용한 데이터 증강(data augmentation)을 적극적으로 적용하였다. 적용된 기법은 다음과 같다.

각 원본 이미지마다 **약 60 개의 변형 이미지**가 생성되었으며, 결과물은 다음과 같은 폴더 구조로 저장되었다.

aug_data/train/
aug_data/val/
aug_data/test/

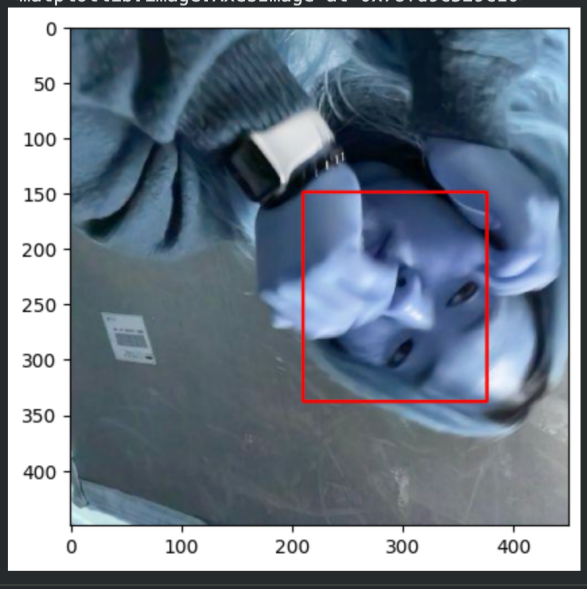
| 변환 기법 | 설명 |
|---|---|
| **RandomCrop** | 임의로 일부를 자르며 부분 얼굴을 모사 |
| **HorizontalFlip / VerticalFlip** | 좌우 혹은 상하 반전을 통한 시각적 다양성 확보 |
| **RandomBrightnessContrast** | 밝기와 대비 변화 |
| **RandomGamma** | 감마 보정을 통한 노출 변화 |
| **RGBShift** | 색상 채널 이동을 통한 조명 색상 변화 |

```
[36]
✓ 1s
import albumentations as alb
augmentor = alb.Compose([alb.RandomCrop(width=450, height=450),
                         alb.HorizontalFlip(p=0.5),
                         alb.RandomBrightnessContrast(p=0.2),
                         alb.RandomGamma(p=0.2),
                         alb.RGBShift(p=0.2),
                         alb.VerticalFlip(p=0.5)],
                        bbox_params=alb.BboxParams(format='albumentations',
                                                   label_fields=['class_labels']))
```

최종적으로 **1,000 장 이상의 학습 데이터셋**을 구성하였다. 이 과정에서 bounding box 좌표 역시 이미지 변환에 맞게 함께 변환되도록 설정하였다.

```
[78]
✓ 0s
cv2.rectangle(augmented['image'],
              tuple(np.multiply(augmented['bboxes'][0][:2], [450,450]).astype(int))
              tuple(np.multiply(augmented['bboxes'][0][2:], [450,450]).astype(int))
              (250,0,0), 2)
plt.imshow(augmented['image'])

<matplotlib.image.AxesImage at 0x78fd9c529c10>
```



### 2.3 모델 구조

모델은 **VGG16 을 backbone 으로 사용하는 CNN 기반 멀티헤드 네트워크**로 구성하였다. 입력 이미지는 120×120 해상도로 리사이즈되며, backbone 을 통과한 feature map 은 Global Max Pooling 을 통해 하나의 feature vector 로 변환된다.

이후 두 개의 출력 헤드로 분기된다.

- **Classification Head**:
  - Sigmoid 활성화를 사용하는 이진 분류
  - 얼굴 존재 여부(face / non-face) 예측
- **Regression Head**:
  - Bounding box 좌표 ($x_1$, $y_1$, $x_2$, $y_2$) 예측
  - Sigmoid 활성화를 사용하여 정규화된 좌표 출력

```
[102]
✓ 0s
def build_model():
  input_layer = Input(shape=(120,120,3))

  vgg = VGG16(include_top=False)(input_layer)

  f = GlobalMaxPooling2D()(vgg)

  # classification model
  class1 = Dense(2048, activation='relu')(f)
  class2 = Dense(1, activation='sigmoid')(class1)

  # bouding box
  regress1 = Dense(2048, activation='relu')(f)
  regress2 = Dense(4, activation='sigmoid')(regress1)

  facetracker = Model(inputs=input_layer, outputs=[class2, regress2])
  return facetracker
```

이 구조를 통해 하나의 모델에서 얼굴 검출과 분류를 동시에 수행할 수 있도록 설계하였다.

### 2.4 손실 함수 및 학습 전략

학습 과정에서는 두 가지 손실 함수를 사용하였다.

- **Classification Loss**: Binary Cross-Entropy
- **Bounding Box Regression Loss**: 좌표 차이 및 박스 크기 차이를 고려한 커스텀 loss

최종 손실 함수는 다음과 같이 정의하였다.

$$\text{Total Loss} = \text{Bounding Box Loss} + 0.5 \times \text{Classification Loss}$$

Optimizer 로는 Adam 을 사용하였으며, 학습 안정성을 위해 learning rate decay 를 적용하였다. 모델은 총 5 epoch 동안 학습되었다.

```
[120]
40m
import tensorflow as tf

logdir = "logs"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)

with tf.device("/CPU:0"):
    hist = model.fit(
        train,
        validation_data=val,
        epochs=5,
        callbacks=[tensorboard_callback]
    )
```

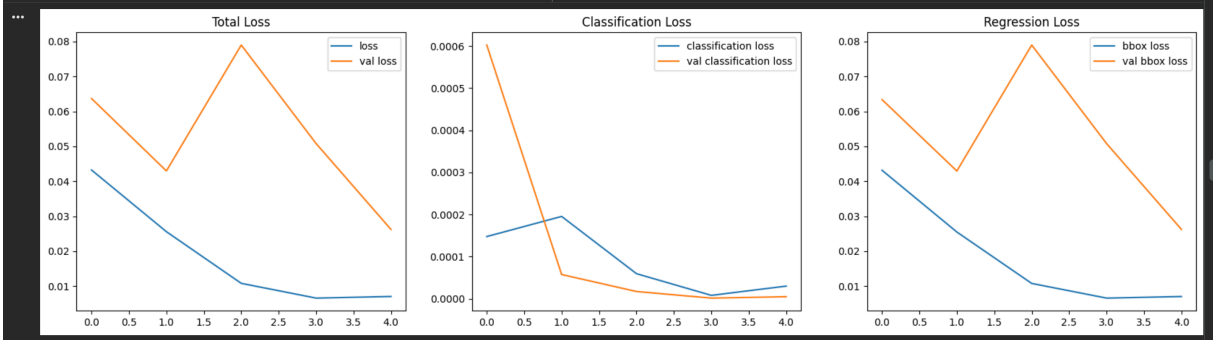
```
Epoch 1/5
135/135 ━━━━━━━━━━━━━━━━━━━━ 468s 3s/step - bbox_loss: 0.1999 - class_loss: 0.0240 - loss: 0.2119 - val_bbox_loss: 0.0634 - val_class_loss: 6.0229e-04 - va
Epoch 2/5
135/135 ━━━━━━━━━━━━━━━━━━━━ 488s 4s/step - bbox_loss: 0.0238 - class_loss: 2.0925e-04 - loss: 0.0239 - val_bbox_loss: 0.0429 - val_class_loss: 5.7618e-05
Epoch 3/5
135/135 ━━━━━━━━━━━━━━━━━━━━ 497s 4s/step - bbox_loss: 0.0142 - class_loss: 7.4525e-05 - loss: 0.0142 - val_bbox_loss: 0.0790 - val_class_loss: 1.7335e-05
Epoch 4/5
135/135 ━━━━━━━━━━━━━━━━━━━━ 494s 4s/step - bbox_loss: 0.0082 - class_loss: 3.1179e-05 - loss: 0.0082 - val_bbox_loss: 0.0508 - val_class_loss: 1.6723e-06
Epoch 5/5
135/135 ━━━━━━━━━━━━━━━━━━━━ 497s 4s/step - bbox_loss: 0.0066 - class_loss: 1.8562e-05 - loss: 0.0066 - val_bbox_loss: 0.0262 - val_class_loss: 4.9297e-06
```

## 3. 최종 결과 및 해석

```
# CLASSIFICATION LOSS
ax[1].plot(hist.history['class_loss'], label='classification loss')
ax[1].plot(hist.history['val_class_loss'], label='val classification loss')
ax[1].set_title('Classification Loss')
ax[1].legend()

# REGRESSION (BBOX) LOSS
ax[2].plot(hist.history['bbox_loss'], label='bbox loss')
ax[2].plot(hist.history['val_bbox_loss'], label='val bbox loss')
ax[2].set_title('Regression Loss')
ax[2].legend()

plt.show()
```




학습 결과, **classification loss 는 매우 빠르게 감소하며 낮은 값으로 수렴**하는 경향을 보였다. 이는 모델이 얼굴과 비얼굴을 효과적으로 구분하고 있음을 의미한다. Validation classification loss 역시 낮은 수준을 유지하여 과도한 overfitting 은 발생하지 않았음을 확인하였다.

Bounding box regression loss 또한 epoch 이 증가함에 따라 지속적으로 감소하였다. 일부 validation 단계에서 손실 값의 변동이 관찰되었으나, 전체적인 감소 추세를 유지하였다. 이는 데이터 증강으로 인한 이미지 다양성과 validation 데이터의 상대적 소규모 특성에 기인한 것으로 해석된다.

전체적으로 classification 과 regression 두 작업 모두에서 손실이 안정적으로 감소하였으며, 이는 제안한 모델이 얼굴 검출 및 인식 작업을 효과적으로 학습하였음을 보여준다.

## 4. 얼굴 인식

얼굴 인식 단계에서는 FaceNet 구조를 기반으로 한 **InceptionResnetV1 임베딩 모델**을 사용하였다. 얼굴 검출 결과로부터 bounding box 영역을 crop 하고, 160×160 크기로 리사이즈한 뒤 정규화하여 임베딩 모델의 입력으로 사용한다. 모델은 입력된 얼굴 이미지를 **512 차원의 임베딩 벡터**로 변환한다.

학습 단계에서는 웹캠으로 수집한 약 30 장의 사용자 얼굴 이미지로부터 임베딩을 생성하고 이를 참조 데이터로 저장하였다.

인식 단계에서는 입력 얼굴의 임베딩과 저장된 사용자 임베딩 간의 **코사인 유사도(cosine similarity)**를 계산한다. 여러 참조 임베딩 중 최대 유사도 값을 기준으로 다음과 같이 판단한다.

- 유사도 ≥ 임계값(threshold): ME (사용자 본인)
- 유사도 < 임계값: NOT ME (타인)

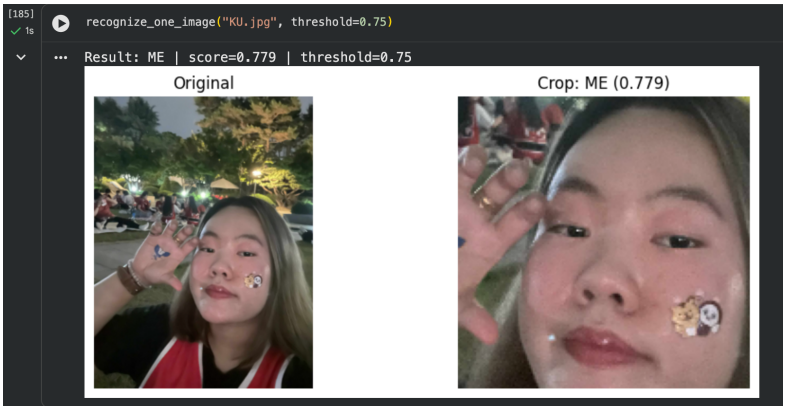



Figure 1. Me

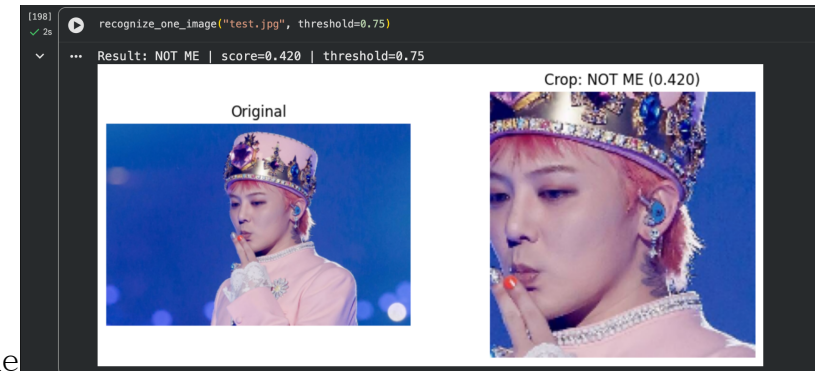



Figure 2. Not me

## 5. 결과 기반 향후 연구 및 개발 가능성

본 프로젝트는 개인 맞춤형 얼굴 검출 및 인식 시스템의 기본 구조를 성공적으로 구현하였다. 그러나 다음과 같은 개선 가능성이 존재한다.

첫째, 더 다양한 사용자 데이터를 포함한 **다중 사용자 얼굴 인식 시스템**으로 확장할 수 있다. 이를 위해 softmax 기반 다중 클래스 분류 구조로의 변경이 가능하다.

둘째, MTCNN 대신 YOLO 또는 RetinaFace 와 같은 최신 얼굴 검출 모델을 적용하여 **실시간 성능 및 검출 정확도**를 향상시킬 수 있다.

셋째, 모바일 또는 임베디드 환경에서의 활용을 위해 **경량 모델(MobileNet, EfficientNet)** 기반 구조로 최적화하는 연구도 가능하다.

마지막으로, 실제 환경에서 발생하는 조명 변화, 가림(occlusion), 다중 얼굴 상황에 대한 강건성을 높이기 위해 데이터셋 확장과 추가적인 augmentation 기법을 적용할 수 있다.