



# 자동 얼굴 블러링 시스템 기획 보고서

## 1. 프로젝트 개요

### 📌 기획 의도

최근 유튜브, 틱톡, 인스타그램과 같은 플랫폼에서 자신의 일상을 영상으로 기록하고 공유하는 사용자들이 급증하고 있습니다. 하지만, 이러한 영상에는 본인의 의도와 관계없이 주변에 있는 제3자의 얼굴이 찍혀 사생활 침해 논란이 발생하고 있습니다.

유튜버나 영상 제작자는 이러한 문제를 해결하기 위해 영상 편집 과정에서 제3자의 얼굴을 모자이크 처리해야 하는데, 이는 많은 시간과 비용이 소요됩니다. 또한, 원치 않게 영상에 찍힌 사람들은 본인의 동의 없이 얼굴이 노출되는 것에 대한 우려를 가지고 있습니다.

본 프로젝트는 이러한 사회적 문제를 해결하기 위해, 영상 촬영 시 "촬영하는 본인"과 "타인"을 구별하여 자동으로 본인이 아닌 얼굴에 블러 효과를 적용하는 AI 기반 얼굴 인식 및 블러링 시스템을 개발하는 것을 목표로 합니다.

### 📌 프로젝트 목표

- YOLO 모델을 활용한 객체 탐지 시스템 구축
- 실시간 웹캠에서 얼굴과 손 탐지
- 브이(V) 손모양이 감지되면 얼굴 블러 해제
- 블러 해제된 얼굴은 지속 유지
- 실시간 영상에서도 빠른 속도로 동작

### 📌 프로젝트 수행을 위해 알아야 할 개념

#### (1) YOLO(Object Detection)

YOLO는 CNN 기반 실시간 객체 탐지 모델이며, 이미지를 한 번만 분석하여 객체를 탐지하는 방식입니다.

얼굴 및 손을 감지하는 데 사용될 수 있으며, 속도가 빠른 것이 특징입니다.

- YOLOv3, YOLOv4, YOLOv5, YOLOv8 등 다양한 버전 존재
- 실시간 영상에서도 **30FPS 이상의 빠른 성능 제공**
- Bounding Box(바운딩 박스)와 객체 클래스 분류를 동시에 수행

## (2) OpenCV

OpenCV는 컴퓨터 비전 라이브러리로, 웹캠 영상 처리 및 이미지 변환 (블러 효과)에 사용됩니다.

- 실시간 영상 캡처( `cv2.VideoCapture` )
- 이미지 처리 및 변환( `cv2.GaussianBlur` )
- YOLO와 결합하여 객체 탐지 가능

## (3) Deep Learning 모델 학습 과정

YOLO 모델을 직접 학습하려면 아래 단계를 거쳐야 합니다.

1. 데이터 수집 (얼굴과 손이 포함된 데이터셋 준비)
2. 데이터 전처리 (라벨링 및 이미지 증강)
3. YOLO 모델 학습 ( `Darknet` 또는 `PyTorch` 활용)
4. 모델 평가 및 튜닝
5. 실시간 웹캠 영상 적용

## 2. 프로젝트 기획

### 주요 기능

1. 사용자 얼굴 인식
  - 사용자의 얼굴을 다양한 각도에서 인식하여 학습
  - 얼굴을 등록한 사용자는 영상 촬영 시 자동으로 식별됨
2. AI 모델 학습

- 전이 학습(Transfer Learning) 기법을 활용하여 사전 학습된 얼굴 인식 모델을 기반으로 사용자 얼굴을 학습
- 딥러닝 및 Yolo, OpenCV를 활용한 얼굴 랜드마크 감지

### 3. 자동 블러링

- 사용자의 얼굴이 아닌 모든 인식된 얼굴에 블러 효과 적용
- 실시간 영상 처리 기능 구현

### 4. 사용자 맞춤형 설정

- 특정 사용자를 화이트리스트에 추가하여 블러 제외 가능
- 

## 기획 기간 및 수행 일정

본 프로젝트의 기획 기간은 4일이며, 아래와 같은 일정으로 진행됩니다.

날짜	수행 내용
1일차	문제 정의 및 기획, 기술 조사, 모델 선정 및 설계
2~3일차	모델 학습 및 구현, 성능 평가 및 테스트, 보고서 작성
4일차	보고서 작성

## 적용 기술

기술 요소	설명
Yolo	실시간 얼굴 감지 및 랜드마크 인식
OpenCV	얼굴 블러링 및 영상 처리
mediapipe	실시간 얼굴 감지 및 랜드마크 인식

## 시스템 동작 흐름

### 1. 사용자 얼굴 등록

- 사용자가 다양한 각도의 얼굴을 입력하여 시스템에 등록
- 모델이 사용자의 얼굴을 학습

### 2. 실시간 얼굴 감지

- 카메라 또는 영상에서 얼굴을 감지

- 사용자의 얼굴과 비교하여 본인 여부 판단

### 3. 자동 블러 처리

- 사용자의 얼굴이 아닌 모든 얼굴에 블러 효과 적용
- ~~실시간 영상 처리 후 저장 또는 스트리밍~~

### 4. 사용자 설정 조정

- 특정 인물 블러 제외
- 특정 동작 블러 제외

## 3. 시스템 구현

### 사전 준비

YOLOv5 설치: <https://docs.ultralytics.com/ko/models/yolov5/>

### 데이터 수집 및 구축

#### (1) 사용할 데이터 종류

- 공개 데이터셋:
  - ~~WIDER FACE Dataset~~ (얼굴 감지 데이터셋)
  - ~~HandPose Dataset~~ (손 감지 데이터셋)
  - **COCO Dataset** (일반 객체 탐지 데이터셋)
  - **WIDER FACE** (얼굴 데이터셋)
- 직접 수집
  - 웹캠을 활용하여 얼굴 및 손 이미지 캡처
  - 휴대폰 영상 촬영 후 프레임 별 이미지 캡처
  - OpenCV를 사용하여 라벨링

#### (2) 데이터 수집 → 약 11,000여 장의 특정 사용자&일반인 얼굴 데이터셋 확보

- 공개 데이터셋 다운로드

1단계 : COCO 데이터셋 활용

<https://cocodataset.org/#download>

- 2017년 Train/Test/Val Images 다운로드
- 2017년 Train/Test/Val Annotations 다운로드
- ▼ 코드 : COCO 2017 데이터셋 중, class가 "person"인 데이터만 다운로드

```
import fiftyone as fo

# COCO 2017 데이터셋을 특정 경로에 다운로드 및 저장
dataset = fo.zoo.load_zoo_dataset(
    "coco-2017",
    split="validation",
    label_types=["detections", "segmentations"],
    classes=["person"],
    # dataset_dir=r"C:\Users\Admin\Desktop\data\mini_project\coco_da
    dataset_name="custom_coco_validation", # 사용자 지정 데이터셋 이름
    overwrite=True # 기존 데이터셋을 덮어쓰도록 설정
)

print("COCO 2017 데이터셋이 지정된 경로에 다운로드 및 저장되었습니다!")
```

## 2단계 : WIDER FACE 얼굴 데이터셋 추가

- COCO 데이터 셋 중 "person"인 데이터만 다운로드 하였으나, "사람"의 데이터 중 "얼굴"이 정확하게 나온 데이터는 상대적으로 적었음
- 본 모델에는 일반 사람의 "얼굴"을 정확하게 인식해야 했으므로, 추가적으로 **WIDER FACE**외부 공개 데이터셋에서 "얼굴" 데이터를 추가
- ▼ 참고) COCO 데이터셋 외 공개 데이터셋

```
# WIDER FACE (얼굴 데이터셋)
wget http://shuoyang1213.me/WIDERFACE/WIDER_train.zip
unzip WIDER_train.zip

# HandPose (손 데이터셋) → 폰 프로젝트에서는 사용 안함
wget https://www.dropbox.com/s/1ue5ohlgfoecfux/handpose_data.
zip
unzip handpose_data.zip
```

### 3단계 : COCO 데이터셋 + WIDER FACE 데이터셋

- 약 5,600여 장의 얼굴 데이터셋 확보

- 직접 수집 (OpenCV로 웹캠 이미지 저장)

#### 1단계 : OpenCV로 웹캠 설정 후 얼굴 이미지 실시간 자동 캡처

- 노트북 웹캠으로 실시간 얼굴 자동 캡처 기능 사용
  - ▼ 코드 : 데이터 수집 및 전처리 - YOLO 로 얼굴 인식 및 라벨링

```
#####  
### 웹캠을 통한 실시간 얼굴 자동 캡처저장 후 YOLO 로 라벨링  
#####  
  
import cv2  
import numpy as np  
import time  
import os  
from pathlib import Path  
from ultralytics import YOLO  
  
# ✅ YOLOv5 모델 로드  
model_path = r'C:\Users\Admin\Desktop\data\mini_project\yolov5r  
model = YOLO(model_path)  
  
# ✅ 웹캠 설정  
cap = cv2.VideoCapture(0)  
  
# ✅ 얼굴 이미지 및 라벨 저장 폴더 생성  
output_dir = os.path.abspath("captured_faces")  
label_dir = os.path.abspath("captured_faces/labels")  
  
Path(output_dir).mkdir(parents=True, exist_ok=True)  
Path(label_dir).mkdir(parents=True, exist_ok=True)  
  
frame_count = 0  
save_interval = 5 # N 프레임마다 저장
```

```

print("🔴 실시간 얼굴 자동 캡처 시작... 'q' 키를 누르면 종료")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    img_h, img_w, _ = frame.shape

    # ✅ YOLOv5를 이용한 얼굴 탐지
    results = model.predict(frame, conf=0.5)

    for result in results:
        boxes = result.boxes.xyxy.cpu().numpy()
        confidences = result.boxes.conf.cpu().numpy()
        classes = result.boxes.cls.cpu().numpy().astype(int)

        for (x1, y1, x2, y2, conf, cls) in zip(boxes[:, 0], boxes[:, 1], boxes[:, 2], boxes[:, 3], boxes[:, 4], boxes[:, 5]):
            x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])

            # ✅ 바운딩 박스 가로폭 줄이기 (좌우 20%씩 축소)
            face_width = x2 - x1
            reduction_ratio_w = 0.2 # 가로폭 20% 줄이기
            new_x1 = x1 + int(reduction_ratio_w * face_width)
            new_x2 = x2 - int(reduction_ratio_w * face_width)

            # ✅ 바운딩 박스 세로폭 줄이기 (상하 15%씩 축소)
            face_height = y2 - y1
            reduction_ratio_h = 0.15 # 세로폭 15% 줄이기
            new_y1 = y1 + int(reduction_ratio_h * face_height)
            new_y2 = y2 - int(reduction_ratio_h * face_height)

            # ✅ 바운딩 박스 표시
            cv2.rectangle(frame, (new_x1, new_y1), (new_x2, new_y2), (0, 255, 0), 2)
            cv2.putText(frame, f'Face: {conf:.2f}', (new_x1, new_y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0))

            # ✅ 자동 캡처 (5프레임마다 저장)
            if frame_count % save_interval == 0:

```

```

timestamp = time.time()
image_filename = os.path.join(output_dir, f"face_{timestamp}")
label_filename = os.path.join(label_dir, f"face_{timestamp}")

# ✅ 저장할 얼굴 크롭 영역도 동일한 바운딩 박스 크기로 설정
face_crop = frame
if face_crop.size != 0:
    cv2.imwrite(image_filename, face_crop)
    print(f"📸 이미지 저장: {image_filename}")

# ✅ YOLO 라벨 데이터 저장 (x_center, y_center, width, height)
x_center = (new_x1 + new_x2) / 2 / img_w
y_center = (new_y1 + new_y2) / 2 / img_h
width = (new_x2 - new_x1) / img_w
height = (new_y2 - new_y1) / img_h

try:
    with open(label_filename, "w") as label_file:
        label_content = f"0 {x_center:.6f} {y_center:.6f} {width:.6f} {height:.6f}\n"
        label_file.write(label_content)

    print(f"📝 라벨 저장 확인: {label_filename}")
    print(f"파일 내용:\n{label_content}")

except Exception as e:
    print(f"❌ 라벨 저장 실패: {e}")

# ✅ 화면 출력
cv2.imshow('YOLOv5 Face Detection', frame)

# ✅ 'q' 키 입력 시 종료
key = cv2.waitKey(1) & 0xFF
if key == ord('q'):
    break

frame_count += 1

```



```
cap.release()
cv2.destroyAllWindows()
```

## 2단계 : 휴대폰 영상으로 촬영 후 프레임별 이미지 캡처

- 노트북 웹캠으로 촬영한 파일에서 조명이나 각도의 다양성 확보 어려움
- 앱 구현을 목표로, 휴대폰 카메라로 얼굴 동영상 촬영 후 프레임별 이미지 캡처 저장
  - ▼ 코드 : OpenCV로 동영상 파일을 프레임별로 이미지 저장

```
#####
### OpenCV로 동영상 파일을 프레임별로 이미지 저장
#####

import cv2
import os

def video_to_frames(video_path, output_folder, image_format='jpg'):
    os.makedirs(output_folder, exist_ok=True)

    cap = cv2.VideoCapture(video_path)
    frame_count = 0

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        frame_filename = os.path.join(output_folder, f'frame_{frame_count}.jpg')
        cv2.imwrite(frame_filename, frame)
        frame_count += 1

    cap.release()
    print(f'{frame_count} frames extracted and saved to {output_folder}')

# 동영상 파일 경로 설정
video_paths = [
    r"C:\Users\Admin\Desktop\data\mini_project\20250122_1.mp4",
    r"C:\Users\Admin\Desktop\data\mini_project\20250122_2.mp4"
```

```
]

# 각 동영상 파일에 대한 프레임 저장
for video_path in video_paths:
    output_folder = os.path.join(os.path.dirname(video_path), f"frame_{video_path}")
    video_to_frames(video_path, output_folder)
```

### 3단계 : 웹캠 이미지 + 휴대폰 촬영 이미지 데이터셋

- 약 6,000여 장의 사용자 얼굴 데이터셋 확보

## 데이터 학습 및 YOLO 모델 생성

### (1) YOLO 데이터셋 포맷 변환 : 이미지 라벨링

YOLO는 `.txt` 파일을 사용하여 **Bounding Box 라벨링**을 수행

(각 이미지에 대해 `.txt` 파일이 생성되며, 좌표 정보가 포함됨)

#### 예시 (YOLO 라벨링 포맷)


```
0 0.5 0.5 0.4 0.4 # 클래스 0 (얼굴), 중심(x,y), 너비, 높이
1 0.6 0.7 0.3 0.3 # 클래스 1 (특정 사용자)
```

### YOLO로 얼굴 인식 → 라벨링

▼ 코드 : 수집 및 전처리 1 - 웹캠 이미지 저장 후 YOLOv5로 얼굴 인식 및 라벨링

```
#####
#### 수집방법 1. YOLOv5로 얼굴 인식 및 라벨링
#####

import cv2
import numpy as np
import time
import os
from pathlib import Path
from ultralytics import YOLO

#  YOLOv5 모델 로드
model_path = r'C:\Users\Admin\Desktop\data\mini_project\yolov5nu.pt'
```

```

model = YOLO(model_path)

# ✅ 웹캠 설정
cap = cv2.VideoCapture(0)

# ✅ 얼굴 이미지 및 라벨 저장 폴더 생성
output_dir = os.path.abspath("captured_faces")
label_dir = os.path.abspath("captured_faces/labels")

Path(output_dir).mkdir(parents=True, exist_ok=True)
Path(label_dir).mkdir(parents=True, exist_ok=True)

frame_count = 0
save_interval = 5 # N 프레임마다 저장

print("🔴 실시간 얼굴 자동 캡처 시작... 'q' 키를 누르면 종료")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    img_h, img_w, _ = frame.shape

    # ✅ YOLOv5를 이용한 얼굴 탐지
    results = model.predict(frame, conf=0.5)

    for result in results:
        boxes = result.boxes.xyxy.cpu().numpy()
        confidences = result.boxes.conf.cpu().numpy()
        classes = result.boxes.cls.cpu().numpy().astype(int)

        for (x1, y1, x2, y2, conf, cls) in zip(boxes[:, 0], boxes[:, 1], boxes[:, 2], boxes[:, 3], confidences, classes):
            x1, y1, x2, y2 = map(int, [x1, y1, x2, y2])

            # ✅ 바운딩 박스 가로폭 줄이기 (좌우 20%씩 축소)
            face_width = x2 - x1

```

```

reduction_ratio_w = 0.2 # 가로폭 20% 줄이기
new_x1 = x1 + int(reduction_ratio_w * face_width)
new_x2 = x2 - int(reduction_ratio_w * face_width)

# ✅ 바운딩 박스 세로폭 줄이기 (상하 15%씩 축소)
face_height = y2 - y1
reduction_ratio_h = 0.15 # 세로폭 15% 줄이기
new_y1 = y1 + int(reduction_ratio_h * face_height)
new_y2 = y2 - int(reduction_ratio_h * face_height)

# ✅ 바운딩 박스 표시
cv2.rectangle(frame, (new_x1, new_y1), (new_x2, new_y2), (0, 255, 0), 2)
cv2.putText(frame, f'Face: {conf:.2f}', (new_x1, new_y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

# ✅ 자동 캡처 (5프레임마다 저장)
if frame_count % save_interval == 0:
    timestamp = time.time()
    image_filename = os.path.join(output_dir, f"face_{timestamp:.0f}.jpg")
    label_filename = os.path.join(label_dir, f"face_{timestamp:.0f}.txt")

# ✅ 저장할 얼굴 크롭 영역도 동일한 바운딩 박스 크기로 설정
face_crop = frame
if face_crop.size != 0:
    cv2.imwrite(image_filename, face_crop)
    print(f"📸 이미지 저장: {image_filename}")

# ✅ YOLO 라벨 데이터 저장 (x_center, y_center, width, height)
x_center = (new_x1 + new_x2) / 2 / img_w
y_center = (new_y1 + new_y2) / 2 / img_h
width = (new_x2 - new_x1) / img_w
height = (new_y2 - new_y1) / img_h

try:
    with open(label_filename, "w") as label_file:

```

```

        label_content = f"0 {x_center:.6f} {y_center:.6f} {width:.6f} {height:.6f}\n"
        label_file.write(label_content)

    print(f"📝 라벨 저장 확인: {label_filename}")
    print(f"파일 내용:\n{label_content}")

except Exception as e:
    print(f"❌ 라벨 저장 실패: {e}")

# ✅ 화면 출력
cv2.imshow('YOLOv5 Face Detection', frame)

# ✅ 'q' 키 입력 시 종료
key = cv2.waitKey(1) & 0xFF
if key == ord('q'):
    break

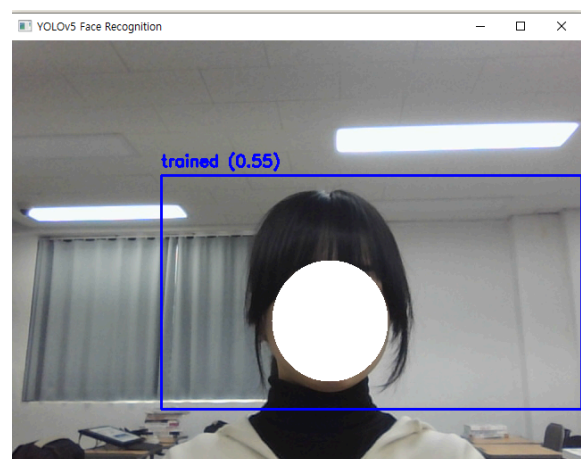
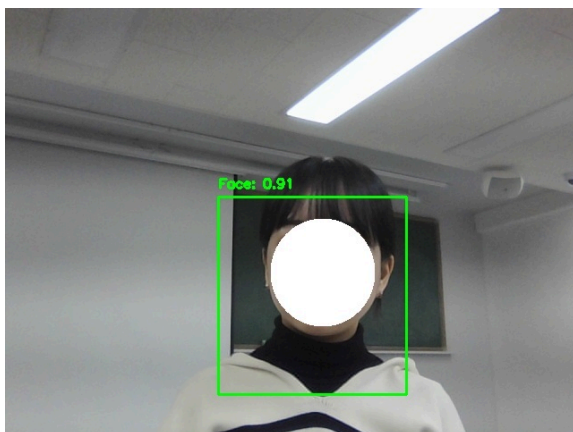
frame_count += 1

cap.release()
cv2.destroyAllWindows()

```

### \*\*\* YOLOv5로 얼굴 인식 시 이슈사항 :

YOLOv5로 "얼굴"을 인식할 때, 바운딩 박스가 크게 잡혀서 모델 학습 후 웹캠으로 얼굴 인식 시 정확도 저하 이슈가 있음



## → 해결 방안 : OpenCV로 얼굴 인식 후 라벨링

▼ 코드 : 특정 사용자 얼굴 이미지 데이터 얼굴 인식 및 라벨링 (1번 클래스)

```
#####
#### 학습 이미지 데이터 : 라벨링
#####

import cv2
import os
import shutil

def detect_faces(image_path, classifier):
    """이미지에서 얼굴을 감지하여 좌표를 반환"""
    image = cv2.imread(image_path)
    height, width, _ = img.shape
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = classifier.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=
    return faces

def process_images(folder_path):
    """폴더 내의 모든 이미지에서 얼굴을 감지하고 개별적인 txt 파일로 저장"""
    classifier = cv2.CascadeClassifier(cv2.data.harcascades + 'haarcasca
    image_files = [f for f in os.listdir(folder_path) if f.endswith(('png', 'jpg', 'j

    for image_file in image_files:
        image_path = os.path.join(folder_path, image_file)
        faces = detect_faces(image_path, classifier)
        txt_filename = os.path.splitext(image_file)[0] + ".txt"
        txt_path = os.path.join(folder_path, txt_filename)

        with open(txt_path, 'w') as f:
            for (x, y, w, h) in faces:
                x_center = (x + w / 2) / width
                y_center = (y + h / 2) / height
                w_norm = w / width
                h_norm = h / height
                f.write(f"1 {x_center:.6f} {y_center:.6f} {w_norm:.6f} {h_norm:.6f}")
```

```

        print(f"{txt_filename} 파일이 생성되었습니다.")

    move_txt_files(folder_path)

def move_txt_files(folder_path):
    """captured_faces 폴더 내의 .txt 파일을 labels 폴더로 이동"""
    labels_folder = os.path.join(folder_path, r"C:\Users\Admin\Desktop\data")
    os.makedirs(labels_folder, exist_ok=True) # labels 폴더가 없으면 생성

    for file in os.listdir(folder_path):
        if file.endswith(".txt"):
            src_path = os.path.join(folder_path, file)
            dst_path = os.path.join(labels_folder, file)
            shutil.move(src_path, dst_path)
            print(f"{file} 이동 완료 -> {dst_path}")

if __name__ == "__main__":
    input_folder = r"C:\Users\Admin\Desktop\data\mini_project\captured_faces"
    process_images(input_folder)

```

▼ 코드 : coco 이미지 데이터 얼굴 인식 및 라벨링 (0번 클래스)

```

#####
#### COCO 이미지 데이터 : 라벨링
#####

import os
import cv2

# Path to dataset
dataset_path = r"C:\Users\Admin\ fiftyone \coco-2017\validation\data"

# Function to detect faces and create YOLO label files
def detect_faces_and_label():
    images_path = os.path.join(dataset_path, r"C:\Users\Admin\ fiftyone \coco-2017\validation\images")
    labels_path = os.path.join(dataset_path, r"C:\Users\Admin\ fiftyone \coco-2017\validation\labels")
    os.makedirs(labels_path, exist_ok=True)

```

```

for image_file in os.listdir(images_path):
    if image_file.endswith((".jpg", ".png", ".jpeg")):
        image_path = os.path.join(images_path, image_file)
        img = cv2.imread(image_path)
        height, width, _ = img.shape

        # Convert to grayscale and detect faces
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        face_cascade = cv2.CascadeClassifier(cv2.data.harcascades + "haarcascade_frontalface_default.xml")
        faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

        # Create label file
        label_file = os.path.join(labels_path, os.path.splitext(image_file)[0] + ".txt")
        with open(label_file, "w") as f:
            for (x, y, w, h) in faces:
                # YOLO format: class_id x_center y_center width height (normalized)
                x_center = (x + w / 2) / width
                y_center = (y + h / 2) / height
                w_norm = w / width
                h_norm = h / height
                f.write(f"0 {x_center:.6f} {y_center:.6f} {w_norm:.6f} {h_norm:.6f}\n")

        print(f"Processed {image_file}, found {len(faces)} face(s).")

# Process
detect_faces_and_label()

print("Labeling completed.")

```

### \*\*\* OpenCV로 얼굴 인식 시 이슈사항 :

사람의 옆 얼굴을 인식하지 못함

→ 해결 방안 : mediapipe로 얼굴 인식 후 라벨링

▼ 코드 : 특정 사용자 얼굴 이미지 데이터 얼굴 인식 및 라벨링 (1번 클래스)



```
#####
#### 학습 이미지 데이터 : 라벨링 (MediaPipe)
#####

import os
import cv2
import mediapipe as mp

# Path to dataset
dataset_path = r"C:\Users\Admin\Desktop\data\mini_project\frames"

# Initialize MediaPipe Face Detection
mp_face_detection = mp.solutions.face_detection
face_detection = mp_face_detection.FaceDetection(model_selection=1,

# Function to detect faces and create YOLO label files
def detect_faces_and_label():
    images_path = os.path.join(dataset_path, r"C:\Users\Admin\Desktop\
    labels_path = os.path.join(dataset_path, r"C:\Users\Admin\Desktop\c
    os.makedirs(labels_path, exist_ok=True)

    for image_file in os.listdir(images_path):
        if image_file.endswith((".jpg", ".png", ".jpeg")):
            image_path = os.path.join(images_path, image_file)
            img = cv2.imread(image_path)
            height, width, _ = img.shape

            # Convert image to RGB for MediaPipe
            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            results = face_detection.process(img_rgb)

            # Create label file
            label_file = os.path.join(labels_path, os.path.splitext(image_file))
            with open(label_file, "w") as f:
                if results.detections:
                    for detection in results.detections:
                        bbox = detection.location_data.relative_bounding_box
                        x, y, w, h = bbox.xmin, bbox.ymin, bbox.width, bbox.heig
```

```

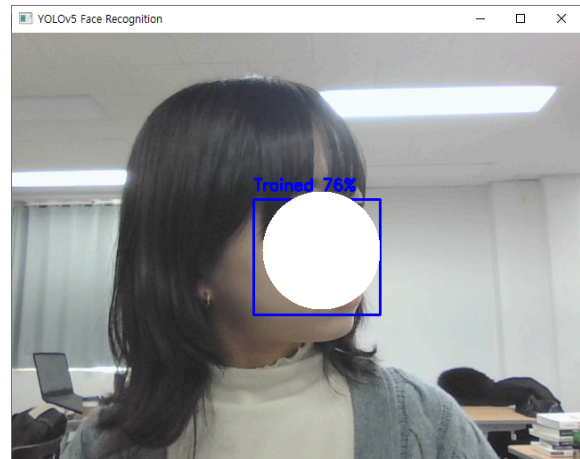
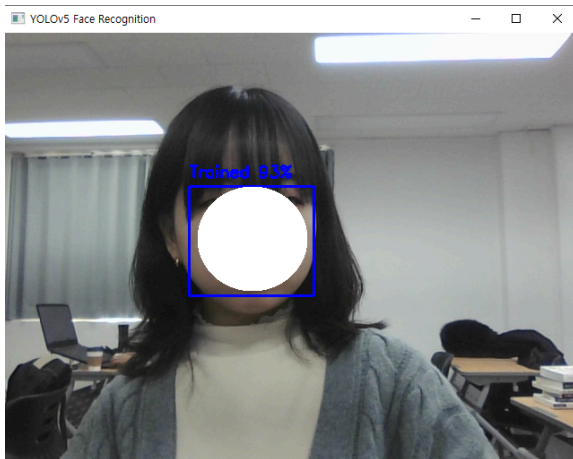
# Convert to YOLO format (normalized)
x_center = x + w / 2
y_center = y + h / 2

f.write(f"1 {x_center:.6f} {y_center:.6f} {w:.6f} {h:.6f}\n")

print(f"Processed {image_file}, found {len(results.detections)} if

# Run face detection and labeling
detect_faces_and_label()
print("Labeling completed.")

```



## (2) Train/Val 데이터셋 분할

- 80% 학습, 20% 검증 데이터로 분할

▼ 코드 : 데이터셋 분할 및 이미지& 라벨링 폴더 이동

```

import os
import shutil
import random

# 🔥 데이터 폴더 설정
image_dir = r"C:\Users\Admin\Desktop\data\mini_project\frames"
label_dir = r"C:\Users\Admin\Desktop\data\mini_project\frames\labels"

```

```

train_img_dir = r"C:\Users\Admin\Desktop\data\mini_project\dataset\in
val_img_dir = r"C:\Users\Admin\Desktop\data\mini_project\dataset\ima
train_label_dir = r"C:\Users\Admin\Desktop\data\mini_project\dataset\l
val_label_dir = r"C:\Users\Admin\Desktop\data\mini_project\dataset\lal

# 🔥 폴더 생성 (이미 존재하면 유지)
os.makedirs(train_img_dir, exist_ok=True)
os.makedirs(val_img_dir, exist_ok=True)
os.makedirs(train_label_dir, exist_ok=True)
os.makedirs(val_label_dir, exist_ok=True)

# 🔥 이미지 리스트 가져오기
images = [f for f in os.listdir(image_dir) if f.endswith('.jpg')]
random.shuffle(images) # 랜덤 섞기

# 🔥 80% 학습, 20% 검증 데이터로 나누기
split_ratio = 0.8
split_index = int(len(images) * split_ratio)

train_images = images[:split_index]
val_images = images[split_index:]

# 🔥 파일 이동 (이미지와 함께 라벨도 이동)
for img in train_images:
    src_img = os.path.join(image_dir, img)
    dst_img = os.path.join(train_img_dir, img)
    shutil.move(src_img, dst_img) # 이미지 이동

# 라벨 이동
label_file = os.path.join(label_dir, img.replace('.jpg', '.txt'))
if os.path.exists(label_file):
    shutil.move(label_file, os.path.join(train_label_dir, os.path.basename(label_file)))

for img in val_images:
    src_img = os.path.join(image_dir, img)
    dst_img = os.path.join(val_img_dir, img)
    shutil.move(src_img, dst_img) # 이미지 이동

```

```

# 라벨 이동
label_file = os.path.join(label_dir, img.replace('.jpg', '.txt'))
if os.path.exists(label_file):
    shutil.move(label_file, os.path.join(val_label_dir, os.path.basename(img)))

print("✅ 데이터셋 분할 및 라벨 이동 완료!")

```

### (3) YOLO 모델 학습

- yaml 파일 생성
  - **YOLO** 등 객체 탐지(Object Detection) 모델에서 사용할 **data.yaml** 파일을 생성
  - 학습/검증 이미지 경로, 클래스 수, 클래스 이름 등 중요한 정보를 **YAML 포맷**으로 저장
    - 0번 클래스 : 일반인 얼굴 / 1번 클래스 : 특정 사용자 얼굴
- ▼ 코드 : 2개 클래스를 학습하여 YOLO 학습용 yaml 파일 생성

```

# 사용자 지정 데이터 경로
train_img_dir = r"C:\Users\Admin\Desktop\data\mini_project\dataset\image\train"
val_img_dir = r"C:\Users\Admin\Desktop\data\mini_project\dataset\image\val"
train_label_dir = r"C:\Users\Admin\Desktop\data\mini_project\dataset\label\train"
val_label_dir = r"C:\Users\Admin\Desktop\data\mini_project\dataset\label\val"

# data.yaml 내용
yaml_content = f"""train: {train_img_dir}
val: {val_img_dir}

nc: 2 # 클래스 개수 (face, user_face)
names: ['face', 'user_face'] # 클래스 이름
"""

# YAML 파일 저장
yaml_path = r"C:\Users\Admin\Desktop\data\mini_project\dataset\data.yaml"
with open(yaml_path, "w") as f:
    f.write(yaml_content)

```

```
print(f"✅ data2.yaml 파일 생성 완료! 위치: {yaml_path}")
```

```
✅ data2.yaml 파일 생성 완료! 위치: C:\Users\Admin\Desktop\data\mini_project\dataset\data2.yaml
```

#### (4) YOLOv5 학습 실행

- Anaconda Prompt → 가상환경 접속
- yolov5 폴더 경로 이동

```
cd C:\Users\Admin\Desktop\data\mini_project\yolov5
```

- yolov5 학습

```
python train.py --img 640 --batch 16 --epochs 32 ^  
--data C:\Users\Admin\Desktop\data\mini_project\dataset_pipeline\data5.y  
aml ^  
--cfg C:\Users\Admin\Desktop\data\mini_project\yolov5\models\yolov5n.ya  
ml ^  
--weights C:\Users\Admin\Desktop\data\mini_project\yolov5\yolov5n.pt ^  
--name yolov5_coco --project C:\Users\Admin\Desktop\data\mini_project\y  
olov5\runs\train
```

##### ▼ 참고 : 명령어 해석

##### 1. python train.py

- YOLOv5에서 제공하는 `train.py` 파이썬 스크립트를 실행합니다.

##### 2. -img 640

- 학습 시 사용할 **입력 이미지 크기**를 지정합니다.
- 640은 한 변의 길이를 의미하며, 일반적으로 이미지가 640×640으로 리사이즈되어 들어갑니다.

##### 3. -batch 16

- \*배치 크기(batch size)\*\*를 설정합니다.
- 한 번의 학습 반복(iteration)마다 몇 개의 이미지를 동시에 처리할지를 정합니다.

- 숫자가 클수록 한 번에 많이 처리하지만, 그만큼 GPU 메모리가 더 필요합니다.

#### 4. **-epochs 32**

- **에폭(epochs)** 수로, 전체 데이터셋을 몇 번 반복하여 학습할지 결정합니다.
- 32라면, 모든 학습 데이터를 32번 반복하여 학습합니다.

#### 5. **-data C:... \data5.yaml**

- 데이터 구성을 설명하는 **YAML 파일**의 경로를 지정합니다.
- 이 파일에는 학습/검증 데이터 경로, 클래스 수( **nc** ), 클래스 이름( **names** ) 등이 기록되어 있습니다.
- 예: `train: [train 이미지 경로], val: [val 이미지 경로], nc: 2, names: [face, user_face]` 등

#### 6. **-cfg C:... \yolov5n.yaml**

- 사용할 **YOLOv5 모델 아키텍처 설정 파일**(.yaml 파일)의 경로입니다.
- 예: `yolov5n.yaml` 은 YOLOv5n(Nano) 버전을 사용하겠다는 뜻으로, 파라미터가 적고 가벼운 모델 구조를 나타냅니다.

#### 7. **-weights C:... \yolov5n.pt**

- 학습에 **\*\*초기 가중치(Pre-trained weights)\*\***를 사용할 파일(.pt)의 경로입니다.
- 여기서는 COCO 데이터셋으로 사전 학습된 `yolov5n.pt` 가중치를 사용합니다.
- 처음부터 학습하는 대신, 이미 학습된 가중치를 이어받아 학습(Transfer Learning)할 수 있습니다.

#### 8. **-name yolov5\_coco**

- 학습 결과를 저장할 **실험 이름**을 정합니다.
- 학습 기록, 모델 가중치, 로그 파일 등이 이 이름으로 폴더가 생성되어 저장됩니다.

#### 9. **-project C:... \runs\train**

- 학습 결과(모델 가중치, 로그 등)를 저장할 **폴더 경로**를 지정합니다.
- 일반적으로 YOLOv5에서는 `runs/train` 아래에 여러 실험이 저장되는데, 여기서는 경로를 직접 지정한 모습입니다.

#### 10. **^ (캐럿 기호)**

- **Windows 명령 프롬프트(CMD)**에서 줄바꿈을 의미합니다.

- 한 줄로 길어지는 명령을 보기 좋게 여러 줄로 구분해주는 역할입니다.
- (Powershell에서는 대신 백슬래시 `\` 가 사용되기도 합니다.)

## 종합 요약

이 명령을 실행하면:

1. 이미지 입력 크기(640×640), 배치 크기(16), 에폭(32) 등의 **학습 파라미터**가 설정됩니다.
2. `data5.yaml` 을 통해 **학습/검증 이미지 경로와 클래스 개수/이름**을 불러옵니다.
3. `yolov5n.yaml` 을 모델 구성 파일로 삼고, **사전 학습된 가중치** `yolov5n.pt` 부터 **Transfer Learning**을 시작합니다.
4. **결과 파일**(학습된 최종 모델, 중간 체크포인트, 텐서보드 로그 등)은 `C:\Users\Admin\Desktop\data\mini_project\yolov5\runs\train\yolov5_coco` 에 저장됩니다.

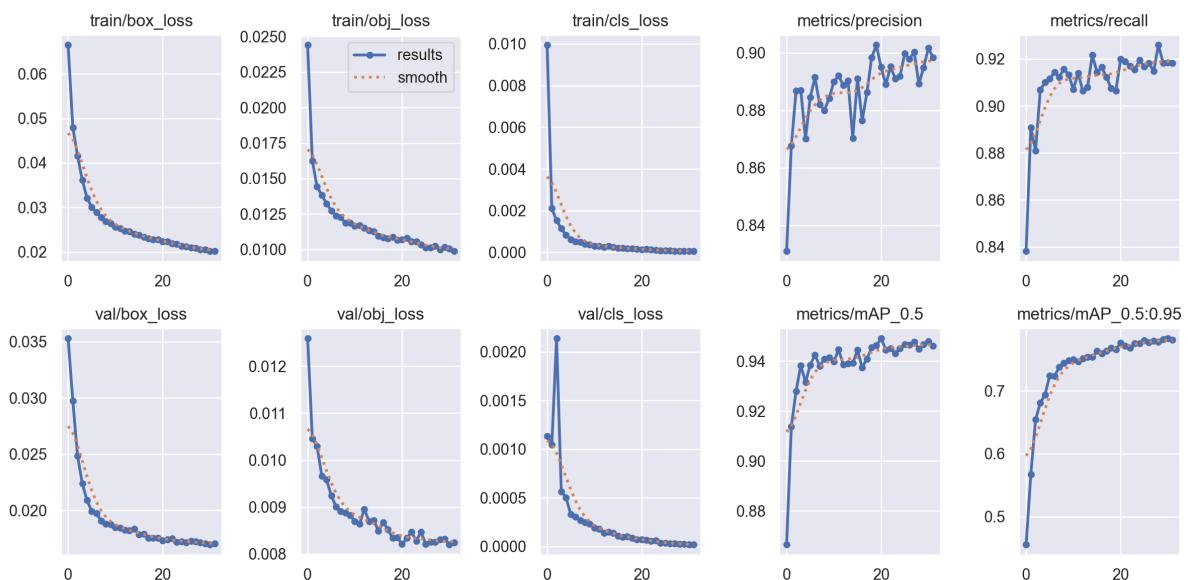
```
Validating C:\Users\Admin\Desktop\data\mini_project\yolov5\runs\train\yolov5_coco16\weights\best.pt...
Fusing layers...
YOLOv5n summary: 157 layers, 1761871 parameters, 0 gradients, 4.1 GFLOPs

```

Class	Images	Instances	P	R	mAP50	mAP50-95
all	1039	583	0.825	0.701	0.761	0.632
face	1039	244	0.762	0.537	0.62	0.453
user_face	1039	339	0.887	0.864	0.902	0.81

```
Results saved to C:\Users\Admin\Desktop\data\mini_project\yolov5\runs\train\yolov5_coco16
```

## (6) YOLOv5 모델 성능 검증



### 1. Train 손실 지표

- **train/box\_loss** : 객체의 바운딩 박스를 얼마나 정확하게 예측하는가를 나타내는 손실
  - 초기 약 0.06대에서 빠르게 떨어져, 최종적으로 0.02 이하까지 감소
    - 바운딩 박스 좌표가 계속해서 실제값과 유사해지고 있다는 의미
- **train/obj\_loss** : 객체가 존재하는지 여부(objectness)에 대한 예측 손실
  - 초기 약 0.025 수준에서 시작해 0.01 내외로 감소
    - 모델이 이미지 내 '객체가 있는지/없는지'를 구분하는 능력이 점차 향상되고 있음
- **train/cls\_loss** : 분류(classification) 손실로, 어떤 클래스인지 정확히 맞추는 정도
  - 초기엔 약 0.01 정도로 꽤 높았다가, 빠르게 0.002 이하 수준까지 감소
    - 클래스 분류가 학습 초기에 비해 상당히 개선되었다는 의미

## 2. Val(검증) 손실 지표

- **val/box\_loss** : 검증셋에 대한 바운딩 박스 예측 손실
  - 초기 0.035 이상에서 시작해 0.02 이하로 점진적으로 감소
  - **train/box\_loss** 곡선과 유사하게 내려가고 있으며, 둘 간의 큰 차이가 보이지 않아 **과적합(overfitting)** 우려는 크지 않음
- **val/obj\_loss** : 검증셋에서의 객체 존재 여부 예측 손실
  - 초기엔 약 0.012 정도로 시작해 0.008 근처까지 완만하게 감소하는 추세
  - train 대비 큰 간극이 없고 안정적으로 줄어들어 검증 성능이 나쁘지 않음
- **val/cls\_loss** : 검증셋에 대한 분류 손실
  - 초반에 약 0.002를 넘다가 학습이 진행됨에 따라 매우 작아짐(0.000x)
  - **train/cls\_loss**와 유사한 양상으로 줄어들고 있어 모델의 일반화 성능이 양호함

## 3. 평가 지표(metrics)

- **precision (정밀도)** : 모델이 예측한 객체 중, 실제로도 객체인 정답의 비율
  - 초기 약 0.85 수준에서 시작해 0.9 이상으로 점차 올라가며, 후반에는 0.90~0.95 사이에서 안정화
  - "검출한 객체들 중 대부분이 실제로 올바른 검출" 임을 의미



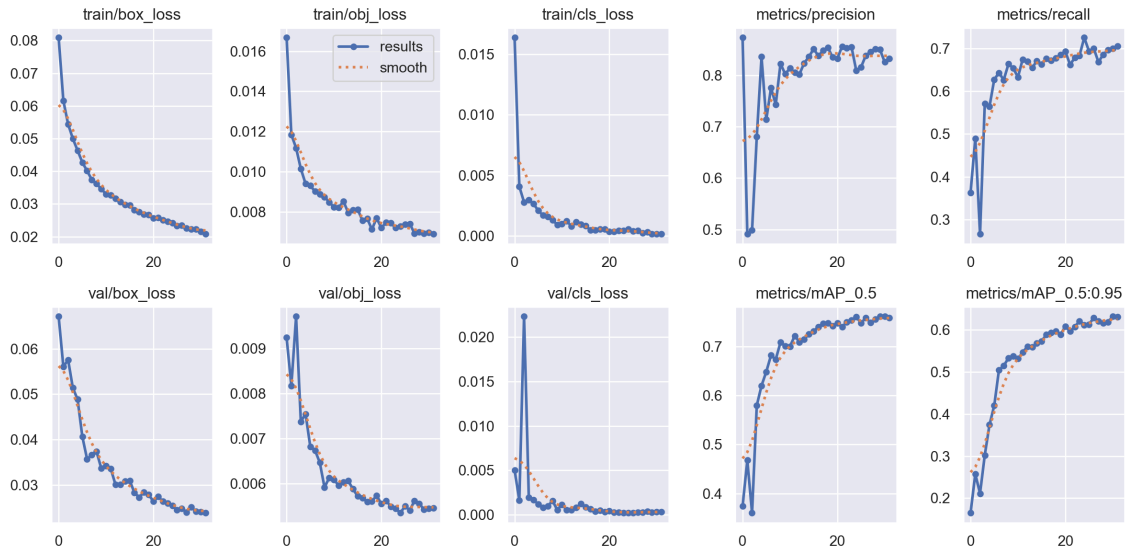
- recall (재현율) : 전체 실제 객체 중 모델이 검출에 성공한 비율
  - 초기 약 0.84 근처에서 시작해 약 0.92 전후로 상승하며, 이후 완만한 증가
  - 모델이 놓치는 객체(검출하지 못한 실제 객체)가 줄어들고 있음
- mAP@0.5 : IOU(Intersection over Union) 임계값이 0.5일 때의 평균 정밀도 (Average Precision)
  - 초기 ~0.5 정도에서 시작해 꾸준히 증가, 최종적으로 0.9 이상에 근접해 높은 정확도
  - 모델이 객체를 인식/검출하는 데 있어서 비교적 관대한(IOU=0.5) 기준으로 매우 높은 성능에 도달
- mAP@0.5:0.95 : IOU 임계값을 0.5부터 0.95까지 0.05 간격으로 적용해 평균을 낸 지표
  - 초기 약 0.40.5 미만 정도에서 시작해 0.70.8대까지 상승
  - 높은 IOU(예: 0.75, 0.85 이상) 조건에서도 모델이 꽤 정확하게 박스 예측
  - Bounding Box 좌표 자체도 정밀하게 맞출 수 있게 되었다는 것을 의미

#### 4. 종합 해석

- 학습 안정성
  - 훈련 손실(Train)과 검증 손실(Val)이 모두 빠르게 감소 후, 비교적 낮은 수준으로 수렴
  - Train과 Val 그래프 간의 큰 차이가 없어, 현 시점에서 **과적합** 문제는 크게 보이지 않음
- 성능 향상
  - Precision, Recall 모두 초반보다 크게 상승하고 0.9 전후를 유지하며, mAP 계열 역시 초기보다 훨씬 높은 값(0.9+/0.75+)에 도달
  - 모델이 점점 **\*\*더 적은 오검출(Precision 상승)\*\***과 **\*\*더 적은 누락(Recall 상승)\*\***을 달성하고, 전반적으로 객체 검출 품질이 뛰어나졌다는 의미
- 실제로 기대할 수 있는 모델 성능
  - mAP@0.5가 0.9 이상이라는 것은, 0.5 이상의 IOU 기준으로는 매우 높은 정확도를 기대
  - mAP@0.5:0.95가 0.7~0.8대라는 것은, IOU 기준을 까다롭게 높여도(0.75, 0.9 등) **바운딩 박스 정밀도 유지**

## ✓ YOLOv5 vs YOLOv8

### ▼ 참고) opencv로 얼굴인식 후 성능 검증



### ▼ 참고) YOLOv5 학습 결과 해석 방법 (Loss & Metrics Curve)

YOLOv5의 학습 결과 그래프로, 각 지표가 에포크(epoch) 별로 변화하는 모습을 보여줍니다.

#### 1. Train Loss (학습 손실)

- **train/box\_loss** (좌측 상단)

- 바운딩 박스(Box) 손실값으로, 예측된 바운딩 박스와 정답 바운딩 박스 간의 차이를 측정
- 초기에는 크지만, 점점 줄어들며 수렴하는 모습 → 모델이 박스 예측을 학습하고 있다는 의미

- **train/obj\_loss** (상단 두 번째)

- 객체(object) 존재 여부에 대한 손실
- 계속 감소하는 모습 → 모델이 객체를 잘 감지하도록 학습되고 있음

- **train/cls\_loss** (상단 세 번째)

- 객체 분류(Classification) 손실
- YOLOv5에서는 클래스 분류보다 객체 감지가 더 중요하지만, 역시 감소하는 모습 → 올바르게 학습됨

#### 2. Validation Loss (검증 손실)

- **val/box\_loss, val/obj\_loss, val/cls\_loss** (하단 1~3번째 그래프)
  - 학습 데이터가 아닌 **검증 데이터**에서 측정한 손실 값
  - **훈련 손실(train loss)**과 유사하게 감소하는 모습이므로 **과적합(overfitting)**이 일어나지 않고 있음
  - 만약 검증 손실이 **증가하거나 변동성이 크다면**, 학습률 조정 필요

### 3. Precision (정밀도) & Recall (재현율)

- **metrics/precision** (상단 네 번째)
  - 모델이 객체를 감지할 때, **정확한 객체만을 감지하는 비율**
  - **0.8에 가까워지며 증가** → 불필요한 오탐(False Positive)이 줄어들고 있음
- **metrics/recall** (상단 다섯 번째)
  - 모델이 실제 객체를 얼마나 놓치지 않고 감지하는지 (재현율)
  - **초기에는 낮지만 점점 증가** → 모델이 더 많은 객체를 정확히 감지하는 방향으로 학습됨

### 4. mAP (Mean Average Precision)

- **metrics/mAP\_0.5** (하단 네 번째)
  - **IoU=0.5**에서의 mAP 값, 즉 정확한 박스가 얼마나 많이 검출되었는지
  - **꾸준히 증가하여 0.6 이상 수렴** → 모델의 성능이 향상되고 있음
- **metrics/mAP\_0.5:0.95** (하단 다섯 번째)
  - **IoU가 0.5~0.95까지 다양한 기준에서의 평균 Precision**
  - **증가하는 경향을 보이며 0.6 근처로 수렴** → 전반적인 객체 탐지 성능이 개선됨

## 결론

### 1. Loss 값 감소

- 훈련( **train** )과 검증( **val** ) 손실이 지속적으로 감소 → 모델이 안정적으로 학습 중
- 과적합(overfitting) 문제 없음

### 2. Precision & Recall 증가

- 정밀도(Precision)와 재현율(Recall)이 함께 증가 → 오탐과 미탐이 줄어들고 있음

### 3. mAP(정확도) 증가

- **mAP@0.5** 와 **mAP@0.5:0.95** 모두 0.6 근처로 수렴
- 객체 탐지 성능이 안정적으로 향상됨

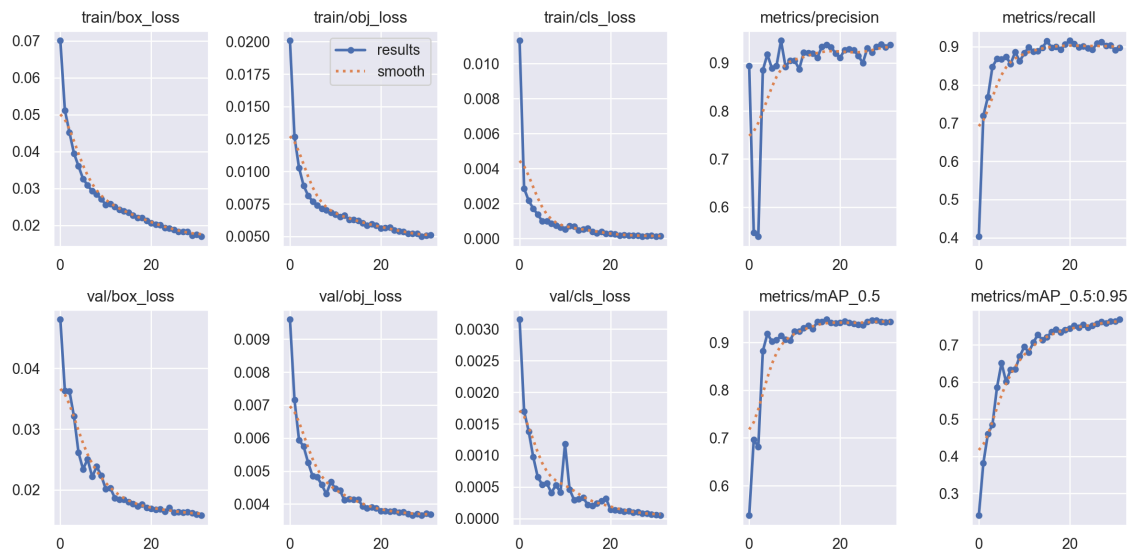
📌 학습이 잘 진행되었으며, 추가적인 튜닝 없이도 좋은 성능을 기대할 수 있음

📌 추가 성능 개선을 원한다면:

- 하이퍼파라미터 튜닝 (학습률, Augmentation 조정)
- 데이터셋 개선 (더 다양한 데이터 추가)
- 에포크(epoch) 증가 (손실값 더 감소할 가능성 있음)

▼ 참고) mediapipe로 얼굴인식 후 성능 검증 → OpenCV랑 비교

```
YOLOv5n summary: 157 layers, 1761871 parameters, 0 gradients, 4.1 GFLOPs
Class      Images  Instances  P      R      mAP50  mAP50-95: 100% | 73/73 [00:16<0
all        2327    3122      0.901  0.919  0.948  0.783
face       2327    1964      0.815  0.843  0.901  0.617
user_face  2327    1158      0.988  0.994  0.995  0.95
Results saved to C:\Users\Admin\Desktop\data\mini_project\yolov5\runs\train\yolov5_coco19
```



### ✓ 1. 손실값 (Loss) 비교

- **train/box\_loss, train/obj\_loss, train/cls\_loss** → 두 결과 모두 점진적으로 감소하며 안정화됨.

- **val/box\_loss, val/obj\_loss, val/cls\_loss** → 초기 손실값이 1번 결과가 더 높았으나, 최종적으로 유사한 수준까지 감소함.
- 차이점:
  - 2번 결과에서 **train/obj\_loss** 값이 더 낮음 → 객체 감지 모델이 더 정확한 예측을 수행함.
  - **val/cls\_loss**가 1번 결과에서 더 요동치는 모습이 보임 → 1번 결과가 클래스 분류에서 약간 불안정.

## ✓ 2. 정밀도 (Precision) 및 재현율 (Recall)

- **metrics/precision** → 2번 결과가 전체적으로 더 높음, 특히 학습 초기부터 더 높은 정밀도를 유지.
- **metrics/recall** → 두 결과 모두 0.9 근처까지 상승하지만, 2번 결과가 초반에 더 빠르게 증가.
- 차이점:
  - 2번 결과가 빠르게 수렴하면서도 최종적으로 더 높은 정밀도를 기록.
  - 이는 잘못된 탐지를 줄이고, 정확한 탐지 성능을 향상시켰다는 의미.

## ✓ 3. 평균 정확도 (mAP)

- **metrics/mAP\_0.5 & metrics/mAP\_0.5:0.95** → 두 결과 모두 학습이 진행되며 0.7~0.9 수준까지 증가.
- 차이점:
  - 2번 결과가 mAP 0.5 및 0.5:0.95에서 더 높은 값을 기록 → 객체 감지 모델의 전체적인 성능이 개선됨.
  - 학습 초반 수렴 속도도 2번 결과가 더 빠름.

## 결론

- 2번 결과가 더 우수한 성능을 보여줌 (정확도, 손실값, mAP 전반적으로 개선됨).
- 초기 학습 단계에서 더 빠르게 수렴하면서도 최종적으로 더 높은 성능을 기록.
- 객체 감지 모델의 정밀도와 재현율이 개선되어 잘못된 탐지를 줄이고, 탐지 성능을 향상시킴.

## 모델 배포 및 실시간 적용

학습된 YOLO 모델을 사용하여 **실시간 얼굴 감지**를 수행합니다.

---

## 4. 결론

본 프로젝트는 유튜버 및 영상 제작자들이 직면한 사생활 보호 문제를 해결하기 위해 AI 기술을 활용하여 자동으로 얼굴을 감지하고 블러링하는 솔루션을 제공합니다.

~~TensorFlow의 전이 학습 모델과 MediaPipe의 얼굴 감지 기능을 결합하여~~, 실시간으로 얼굴을 분석하고 블러를 적용함으로써 기존의 수작업 편집을 대체할 수 있습니다. 이를 통해 영상 제작자의 시간과 비용을 절감할 뿐만 아니라, 원치 않게 영상에 노출되는 사람들의 사생활 보호에도 기여할 수 있습니다.

향후 발전 방향으로서는 추가적인 AI 학습을 통한 인물 식별 정밀도 향상, GPU 가속을 통한 실시간 성능 최적화, 그리고 클라우드 기반 서비스로의 확장을 고려할 수 있습니다.