공학석사학위논문

객체 검출 모델을 이용한 SMT 솔더 페이스트 도포 결함 검출

Defect Detection of Solder Paste Printing in SMT Process Using Object Detection Models

충 북 대 학 교 대 학 원

산업인공지능학과 산업인공지능학전공

김 혜 영

2025년 8월

공학석사학위논문

객체 검출 모델을 이용한 SMT 솔더 페이스트 도포 결함 검출

Defect Detection of Solder Paste Printing in SMT Process Using Object Detection Models

지 도 교 수 전 명 근

산업인공지능학과 산업인공지능학전공

김 혜 영

이 논문을 공학석사 학위논문으로 제출함.

2025년 8 월

본 논문을 김혜영의 공학석사학위 논문으로 인정함.

심 사 위 원 장　　박 태 형 ㊞

심 사 위 원　　전 명 근 ㊞

심 사 위 원　　류 관 희 ㊞

충 북 대 학 교 대 학 원

2025년 8월

차 례

Defect Detection of Solder Paste Printing in SMT Process Using Object Detection Models

Kim, Hye Young

Department of Industrial Artificial Intelligence
Graduate School, Chungbuk National University
Cheongju, Korea
Supervised by Professor Chun, Myung Geun

Abstract

This study investigates real-time detection of solder paste defects on PCB surfaces in SMT processes by comparing two state-of-the-art object detection models: YOLOv10 and RT-DETR. Using 6,724 high-resolution images collected from actual production lines, both models were trained under identical conditions. YOLOv10 demonstrated advantages in speed and efficiency, while RT-DETR achieved higher accuracy (mAP@0.5: 0.809), further improved to 0.973 with RandAugment. These results highlight the potential of deep learning-based inspection systems as low-cost alternatives to SPI equipment, with applicability to embedded real-time SMT inspection.

key words : SMT process, Solder paste defects, Object detection, RT-DETR, YOLOv10

---

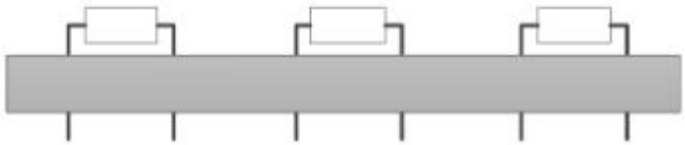
* A thesis for the degree of Master in August 2025.

표 차례

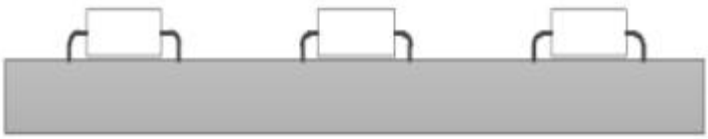
그림 차례

Ⅰ. 서 론

전통적인 관통형 실장 기술(THT)은 <그림 1>과 같이 부품의 핀을 인쇄회로기판(PCB)에 뚫린 구멍을 통해 삽입한 뒤 납땜하는 방식이다. 모든 부품이 핀 형태로 설계되어야 하므로 부품의 크기를 작게 만들기 어려워 회로기판의 소형화에 한계가 있다. 이에 비해 표면 실장 기술(SMT)[1]은 PCB에 구멍을 뚫지 않고, 표면실장소자(SMD)를 PCB 위의 납땜 패드에 고정하는 기술로 SMT에서는 부품의 다리에 솔더 페이스트(solder paste)를 도포한 뒤 기판 위에 부착하여 납땜을 수행한다. 이로 인해 부품 다리의 길이를 줄일 수 있어 결과적으로 <그림 2>와 같이 제품을 소형화할 수 있는 장점이 있다.



<그림 1> THT 기법 [2]



<그림 2> SMT 기법 [2]

하지만, 대량 생산을 할 때 이 과정에서 납땜 관련 불량이 많이 발생하는데 PCB 위의 SMD가 조밀하게 배치되고 접합부가 매우 소형화되어 있어 검사가 어렵다. 작은 결함 하나가 PCB의 기능에 큰 영향을 미칠 수 있으므로 생산 초기에 불량을 검출하는 것이 매우 중요하다. SMT 공정의 납땜 접합부 검사의 중요성에 관해서는 다양한 연구가 있다[3].

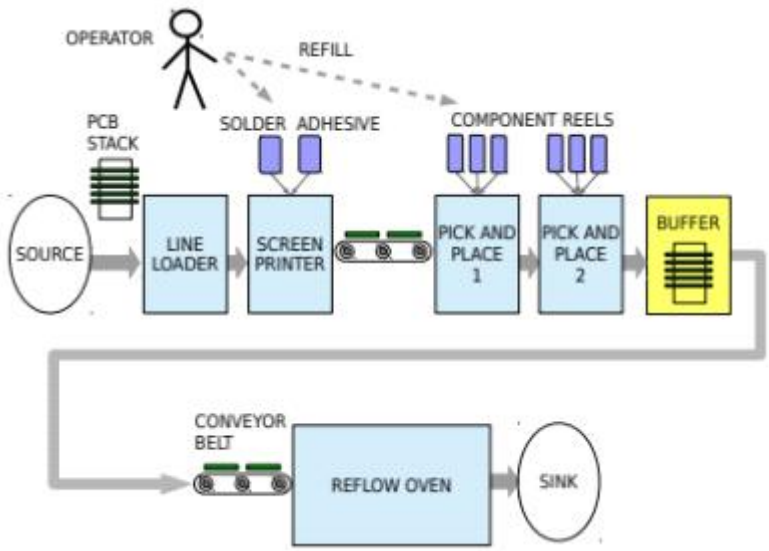
SMT 공정은 크게 네 단계로 구분되며 전체적인 조립 라인은 <그림 3>과 같다.

1) 솔더 페이스트 도포 및 검사: 스텐실(stencil) 또는 스크린 프린터(screen printer)를 사용해 PCB에 솔더 페이스트를 인쇄하며, 프린터마다 PCB 형상에 맞는 프레임을 사용한다[1]. 솔더 페이스트 검사(SPI) 장비로 솔더 페이스트의 형상, 처짐(slump), 점도 등을 측정하여 품질을 검사한다.

2) 부품 실장: 픽 앤 플레이스(pick and place) 장비로 전자 부품을 납땜 패드 위에 배치한 뒤, 작업자가 육안으로 실장 오류를 간단히 확인한다.

3) 리플로우 납땜: PCB를 리플로우 오븐에 통과시켜 단계적으로 가열 및 냉각함으로써 솔더가 고체화되고 부품이 영구 부착된다. 이때 이송 벨트 위에서 플럭스 잔여물과 오염 물질을 제거하기 위해 용매 세척기를 사용한다.

4) 자동 광학 검사(AOI): 실장과 납땜 접합부 모두를 AOI 장비로 검사하며, 이를 통해 배치 오류와 납땜 불량을 모두 탐지한다[1],[4].



<그림 3> SMT-PCB 조립 라인 구성 [5]

여러 문헌과 기술 보고서에 따르면, SMT PCB 제조 불량의 약 70%가 솔더 페이스트 인쇄 공정에서 발생하는 것으로 보고되었다[6-8]. 따라서 솔더 페이스트 공정에서의 품질 관리는 PCB의 장기 신뢰성을 보장하는 데 매우 중요하다. 게다가, 리플로우 이전 단계에서 발생하는 불량 비용은 리플로우 이후 불량 비용 대비 10배 저렴하며, 회로 검사 단계 불량 대비는 70배, 현장(필드) 불량 대비로는 700배 더 적게 소요되는 것으로 계산된 바 있다. 이처럼 솔더 페이스트 검사(SPI)는 납땜 불량을 최소화하고 제조 비용을 절감하기 위한 필수적인 공정이라고 할 수 있다.

PCB 결함 검출 방법으로는 기술자의 주관적 판단에 의존하는 수작업 검사, PCB 기능을 테스트하여 결함 유무를 확인하는 성능 시험, 기준이 되는 정밀한 템플릿(원본 PCB 이미지)과 검사 대상 PCB를 매칭하여 결함 유무를 판별하는 기준 비교 방법[9], 그리고 앞서 언급한 자동 광학 검사(AOI)[10]가 있다. 이와 같은 전통적 검사 방법과 비교했을 때, 딥러닝 기반 결함 검출 방법은 더 높은 성능을 보이며 낮은 정확도와 비효율성을 효과적으로 개선할 수 있어 최근 몇 년간 딥러닝 기반 PCB 결함 검출이 핵심 연구 주제로 떠올랐다. Li 등[11]은 실제 PCB 이미지와 가상의 PCB 이미지를 결합한 데이터 전처리 기법을 적용한 후 YOLOv3의 예측 출력 계층을 3단계에서 4단계로 확장함으로써 93.07%의 검출 정확도를 달성했다. 또, Hu 등[12]은 Faster R-CNN을 개선하고 ResNet50 기반의 이미지 피라미드(feature pyramid) 네트워크를 특징 추출기로 활용하여 일반적인 PCB 결함 유형을 검출한 연구가 있다. 실제 SMT 생산 라인에서 핵심 공정인 솔더 페이스트 도포 단계에서 불량을 실시간으로 정밀 검출하기 위해 딥러닝에 기반한 특화된 연구는 아직 부족한 실정

이다.

YOLO는 경량화된 네트워크 구조와 높은 처리 속도를 바탕으로 실시간 검출에 유리하지만, 복잡한 장면에서 작은 결함을 놓치기 쉽고 앵커 박스 설정에 민감하다는 한계가 있다. 반면, RT-DETR은 Transformer 기반의 전역적 특징 학습으로 앵커 박스 없이도 다양한 크기의 객체를 안정적으로 검출할 수 있으나 학습 초기 수렴 속도가 느리고 연산량이 상대적으로 크다는 문제가 있다.

본 연구에서는 YOLOv10과 RT-DETR을 실제 데이터에 적용하여 SMT 공정 중 발생하는 솔더 페이스트 도포 불량을 검출하는 성능을 비교 평가하였다. 이를 통해 실시간 응용에 적합하면서도 높은 정확도를 보이는 최적의 딥러닝 기반 객체 검출 모델을 도출하고자 한다. 더불어 최종적으로는 고가의 SPI(Solder Paste Inspection) 장비 도입이 어려운 중소 SMT 제조 기업에서 SPI 기능을 지원하는 실시간 검사 모듈 개발을 위한 선행 연구 자료로 활용하는 것을 목적으로 한다.

Ⅱ. PCB 결함 검출을 위한 기존 기법과 딥러닝 모델 비교 관련 연구

2.1. AOI 기반 PCB 결함 검출

AOI(Automated Optical Inspection) 기반 PCB 결함 검출 영역은 크게 임계값/이진화 기반, 에지/형태분석 기반, 템플릿 매칭 기반으로 구분되며, 본 절에서는 각 기법의 주요 연구 사례를 검토한다.

2.1.1 임계값/이진화 기반

AOI 기반 PCB 결함 검출 영역은 크게 임계값/이진화 기반, 에지/형태분석 기반, 템플릿 매칭 기반으로 구분되며, 본 절에서는 각 기법의 주요 연구 사례를 검토한다.

기준 PCB 이미지와 검사 대상 PCB 이미지 간의 픽셀 차이를 추출하여 임계값 이진화로 결함 영역을 분리하는 기법에는 두 가지 대표적 접근이 있다.

첫째, 단순 이미지 뺄셈 방식(Subtraction Method)[13]은 양품 PCB 한 장을 기준으로 검사 영상을 뺄셈한 후(diff = |standard - target|), 이 값을 0과 비0으로 이진화(mask = diff > 0)하여 잠재적 결함 영역을 도출한다. 이후 particle analysis를 통해 크기·형태 기반 필터링을 수행하므로 구현이 간단하고 연산 비용이 낮아 실시간 검사에 적합하다. 다만, 조명 변화나 카메라 노이즈에 취약하고 결함 종류 분류에는 추가 로직이 필요하다는 한계가 있다.

둘째, 동적 기준 이미지 갱신 및 개선된 영역 성장 기법은 다수의 양품 이미지를 평균화하여 기준 이미지를 생성하고, 검사 중에도 이를 동적으로 업데

이트함으로써 영상의 안정성을 확보한다[14]. Bilateral Filtering으로 노이즈를 억제한 뒤 히스토그램 기반 임계값 산출을 통해 이진화를 수행하고, 개선된 Region Growing을 적용하여 그레이 값 변화와 경계 길이를 동시에 고려해 결함 영역을 완전하게 분리한다. 이 방식은 조명·위치 편차에 강인하여 누락 없는 검출이 가능하나, 평균화·갱신·필터링·영역 성장 절차로 인해 연산량이 증가한다[14].

2.1.2 에지/형태분석 기반

원형 홀 등 특정 기하학적 구조를 검출하기 위해, 적응형 에지 검출과 거리 변환(distance transform)을 활용한 기법[15]이 제안되었다. 대표적으로 A Fast PCB Hole Detection Method Based on Geometric Features에서는 적응형 에지 검출로 에지 맵을 생성한 후 거리 변환을 통해 각 픽셀의 유클리드 거리를 계산한다. 이어서 Rapid Circle Center Search(RCCS) 알고리즘으로 원 중심 후보를 스캔하고, 하우스도르프 거리(Hausdorff distance)를 이용해 거짓 양성을 걸러낸다. 합성 영상 및 실제 PCB(OSHPark 데이터셋) 테스트에서 평균 실행 시간 1.34 s, F-measure 0.9944를 기록하며 Atherton's CHT 대비 속도와 정확도 모두 우수함을 입증하였다. 또한, Th = 8 범위에서 에지 폐기 임계값을 최적화함으로써 처리 효율을 극대화하였다.

2.1.3 템플릿 매칭 기반

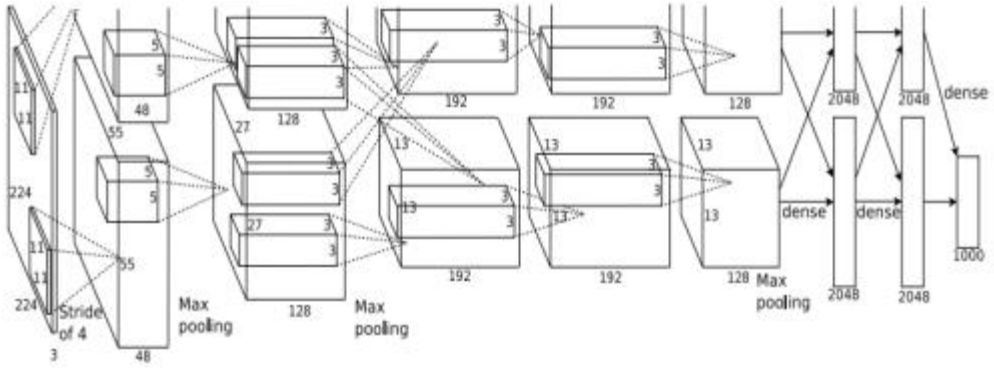
템플릿 매칭 방식[16]에서는 정규화 상호상관(NCC)을 이용해 기준 패턴과 검사 영상 간 유사도를 계산하나, 계산 비용이 높아 실시간 적용이 어려웠다.

Improved Normalized Cross-Correlation(INCC) 기법은 2D 서브이미지를 1D 특징 디스크립터로 변환하고, 디스크립터에 픽셀 중복 기여 및 통계치를 결합한 뒤 DCT 변환을 거쳐 주파수 영역에서 NCC 연산을 수행한다. 실제 PCB 32장 데이터셋에서 기존 NCC와 동등한 정밀도(Precision, Recall, F-measure)를 유지하면서 연산 시간을 대폭 단축하였으며, 노이즈 강인성도 확보하였다.

전통적인 AOI 기반 PCB 결함 검출 기법은 연산 비용이 낮고, 구현이 용이하다는 장점이 있지만, 조명·환경 변화에 민감하고 복합적인 결함을 정확히 인식하거나 실시간 대응을 하기에는 한계가 있다. 이러한 한계를 극복하기 위해 최근 딥러닝 기반 접근법들이 주목받는 추세다. 다음 절에서는 딥러닝 기반 기법 중 특히 CNN 및 YOLO 계열 구조를 중심으로 기존 AOI 방식과의 차별점과 성능 향상 가능성을 비교·분석한다.

2.2. CNN 기반 PCB 결함 검출

PCB 결함 검출에서 딥러닝 기반 접근은 숙련된 작업자가 필요한 인력 문제나 기존 AOI 방식의 한계를 해결하기 위한 대안으로 주목받고 있다. 특히, CNN(Convolutional Neural Network)을 활용한 방식은 이미지로부터 특징을 자동으로 추출하고 학습하는 구조로 다양한 형태의 결함에 대한 높은 일반화 성능과 정확도 향상이 기대되는 모델이다. 이러한 CNN 모델 중 하나로 대표적인 구조는 AlexNet이 있으며, 이는 초기 딥러닝 기반 이미지 분류 모델로서 여러 분야에서 널리 활용되어 왔다.



<그림 4> CNN(Alexnet) 아키텍처 예시 [17]

<그림 4>는 AlexNet의 전체적인 구조를 나타낸 것으로, 입력 이미지에 대해 다층의 Convolutional Layer와 Max Pooling Layer를 거쳐 점차 고수준의 특징을 추출한 후, Fully Connected Layer를 통해 최종 분류를 수행하는 방식이다.

연구 [18]에서는 기존 AOI 방식의 한계를 극복하고, 다양한 SMT 제품군에서 발생하는 결함을 검출하기 위해 CNN을 활용한 방식을 적용했다. ResNet, GoogleNet, VGGNet, AlexNet 등의 사전 학습(pre-trained)된 모델을 기반으로 전이 학습(transfer learning)을 적용하여, 학습 시간 단축 및 소량의 불균형 데이터로도 과적합(overfitting)을 방지하며 높은 정확도를 달성했다. 선행 연구 결과들에 따르면, CNN 기반 결함 검출 모델은 최대 99%의 정확도를 기록하며, 기존의 엣지 검출이나 SVM, 랜덤 포레스트, 로지스틱 회귀 등의 머신러닝 기법을 모두 능가하는 성능을 보였다.

한편, SMT 공정의 솔더 페이스트 단계에서 발생하는 여섯 가지 결함 클래스(불가교정 결함, 솔더 누락, 과다 솔더, 단락, 정의되지 않은 객체, 정상 솔

더)를 조기에 검출하기 위해, 2D 신호 처리 기법과 CNN을 결합한 최적화 기반 딥러닝 모델을 제안하였다.

제안된 방법은 먼저 PCB 영상을 전처리하여 솔더 패드 영역을 정확히 분리하고, 대비 조정 및 에지 강조 필터 등 다양한 이미지 프로세싱 기법을 최적화하여 CNN 입력의 품질을 향상시켰다. 이후 CNN 분류기를 학습시켜 여섯 가지 클래스에 대한 분류를 수행하였으며, 테스트 데이터셋(총 648개 샘플)을 기반으로 96.40%의 높은 정확도를 달성하였다.

또한, 네트워크의 성능 향상을 위해 ReLU 활성화 함수와 함께 AdaMax 최적화 기법을 적용하였으며, 이는 RMSprop 및 Adam 등 기존 최적화 방식보다 안정적이고 효율적인 학습 성능을 보였다. 검출된 결함은 원본 PCB 이미지에 프레임으로 표시되어 운영자가 직관적으로 결함 위치를 확인할 수 있도록 하였다.

본 접근 방식은 기존 머신러닝 기반 검출 방식보다 조기 탐지 성능이 우수하며, SMT 라인 내 실시간 품질 검사 시스템에 효과적으로 적용될 수 있는 가능성을 제시한다[19].

2.3. YOLO 기반 PCB 결함 검출

최근에는 Swin Transformer나 ViT 계열의 자기주목(self-attention) 기반 백본을 특징 추출기로 활용하고, 이를 YOLO 계열의 경량 객체 검출기와 결합한 하이브리드 방식이 PCB 결함 검출 분야에 활발히 적용되고 있다. 본 절에서는 이러한 Transformer-YOLO 융합 모델의 설계 요소와 성능 개선 효과를 중점적으로 살펴본다.

PCB 결함 검출의 정확도 및 효율성을 높이기 위해 기존 연구에서는 Transformer와 YOLO를 결합한 하이브리드 네트워크를 제안하였다[20]. 먼저, 기존 YOLO의 앵커 박스 생성을 위한 K-Means 클러스터링을 PCB 결함 크기 분포에 최적화된 개선된 클러스터링 기법으로 대체하여 데이터셋에 적합한 앵커 프라이어를 도출하였다. 이어서 전통적인 컨볼루션 기반 백본 대신 Swin Transformer를 도입함으로써, shifted window self-attention을 통해 지역 및 전역 특징 간의 의존성을 효과적으로 학습하도록 하였다. 마지막으로, 검출 헤드에 채널별 attention과 pointwise convolution을 결합한 경량 모듈을 삽입하여 네트워크가 정보량이 큰 특징 채널에 더욱 집중하도록 하였다.

다섯 차례의 기능 제거 연구(ablation study) 결과, 기본 YOLOv5 대비 앵커 클러스터링 개선만으로 mAP가 0.62%p 향상되었으며, Swin Transformer 백본 적용 시 추가로 5.33%p, attention 모듈 삽입 시 0.60%p가 각각 향상되어, 최종 모델은 97.04%의 mAP를 달성하였다. 이는 Faster R-CNN, SSD, YOLOv3·v4에 비해 최대 23.90%p까지 뛰어난 성능이며, 산업용 AOI 환경에서 요구되는 모델 크기와 추론 속도를 모두 만족한다.

기존의 PCB 관련 연구들은 대부분 SMT 공정의 AOI 단계를 통과한 회로 패턴이나 부품 실장 완료 후의 PCB 이미지를 데이터셋으로 활용하여 결함 검출을 수행하였다. 그러나 서론에서 언급한 바와 같이, 스크린 프린팅 단계에서의 솔더 페이스트 도포 여부 역시 최종 제품의 품질에 결정적인 영향을 미치므로, 본 논문에서는 해당 공정 단계의 데이터를 활용하여 실험을 진행하였다. 특히, 실시간 처리 속도와 연산 리소스 효율성, 그리고 높은 검출 정확도와 정

밀도를 모두 만족시키기 위해, 최신 객체 검출 모델인 YOLOv10과 RT-DETR을 사용하여 비교·분석을 수행하고자 한다.

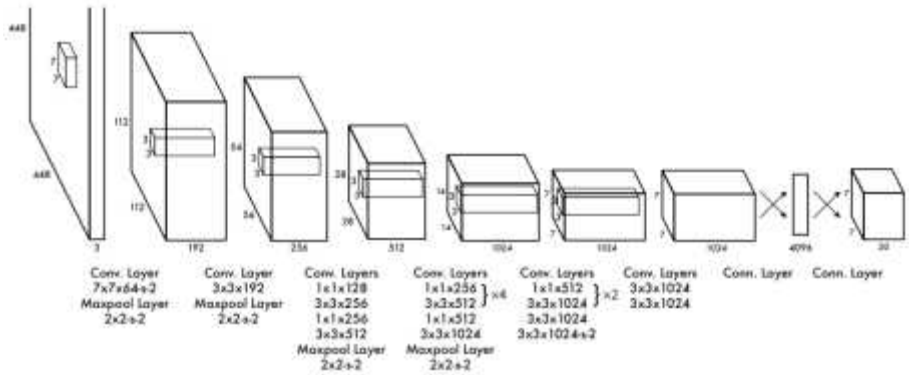
Ⅲ. PCB 솔더 페이스트 도포 결함 검출 연구에 사용된 딥러닝 객체 검출 모델

3.1. YOLOv10의 아키텍처 및 특징

사람은 어떤 이미지가 주어졌을 때, 어떠한 객체가 어디에 위치하는지 빠르게 파악할 수 있다. 빠르고 정확한 객체 검출 알고리즘은 특수한 센서 없이도 컴퓨터가 사람과 같이 실시간으로 장면 정보를 처리 및 전달할 수 있게 한다.

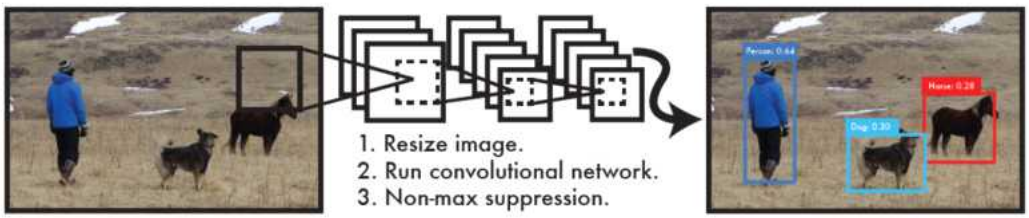
기존에도 여러 객체 검출 시스템이 있었지만, 이들은 이미지의 모든 위치를 반복해서 검사하여 계산량이 매우 많은 단점이 있다[21]. 또, R-CNN, Fast R-CNN, Faster R-CNN과 같은 모델은 여러 구성요소를 순차적으로 연결하는 복잡한 파이프라인을 가지고 있어 학습 최적화가 어려운 단점이 있다[22].

YOLO(You Only Look Once)는 이미지를 한 번만 인식하여 어떤 객체가 어디에 있는지 예측할 수 있는 1-stage 객체 검출 모델이며, 아키텍처는 <그림 5>와 같다. 20개의 컨볼루션 계층 뒤 평균 풀링(average-pooling) 계층과 완전 연결(fully connected) 계층을 붙인 구조를 사용한다.



<그림 5> 기존 YOLO Architecture [23]

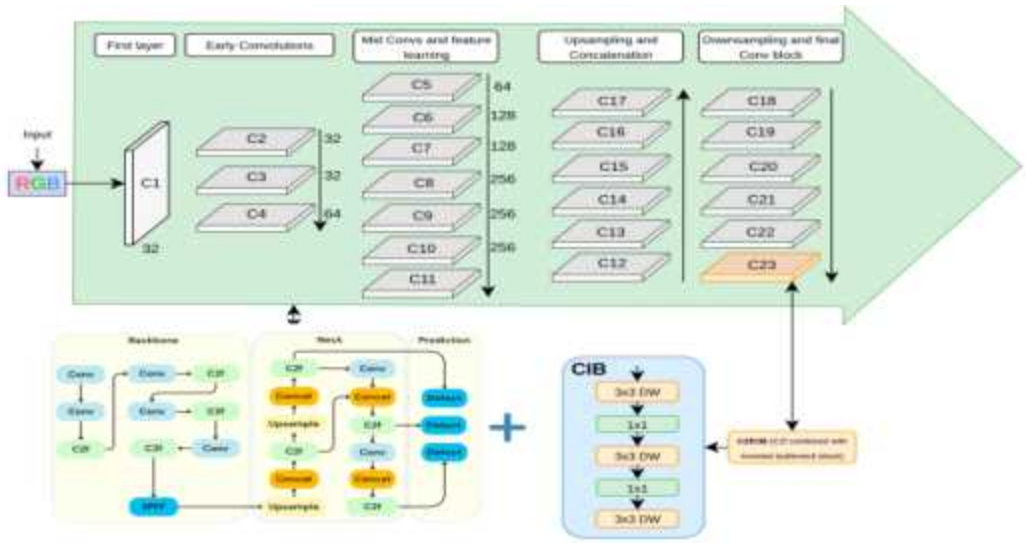
검출을 회귀 문제로 다루기 때문에 복잡한 파이프라인이 필요 없으며 <그림 6>과 같이 하나의 컨볼루션 신경망이 동시에 여러 바운딩 박스와 각 박스의 클래스 확률을 예측한다. 전체 이미지를 기반으로 빠르게 학습 및 추론하여 검출 성능(평균 정밀도, mAP)을 직접 최적화하는 특징을 가진다.



<그림 6> The YOLO Detection System [23]

YOLOv10은 진정한 End-to-End 검출을 위해 NMS(Non-maximal Suppression) 단계를 제거하고, 블록별 연산 효율을 극대화하는 구조적 개선을 결합한 모델이다. 전체 아키텍처는 <그림 7>과 같다. 상단 패널은 RGB 입력을 CSPNet 기반 Backbone(C1 - C4)을 통해 P3 - P5 스케일의 특징 맵으로

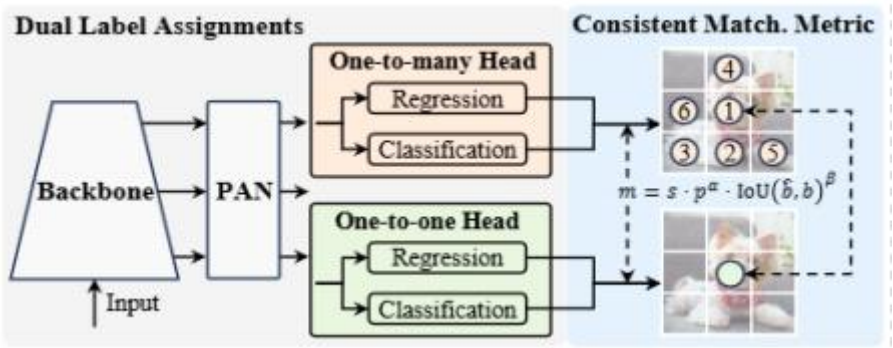
변환한 뒤, PAN(Neck)에서 멀티스케일 피처를 결합하는 과정을 보여 준다. 하단 좌측 패널은 Backbone - Neck - Prediction(Detection Head) 전체 흐름을 요약하며, 우측 “CIB(Compact Inverted Block)” 박스는 공간-채널 분리형 다운샘플링 구조를 상세히 확대한 것이다. 여기서 3×3 DW(Depthwise) 레이어는 공간 정보를, 1×1 Pointwise 레이어는 채널 정보를 각각 전담 처리하여 불필요한 연산 중복을 제거한다. 또한, Prediction 헤드의 1×1 Conv 계층은 채널 수를 크게 축소하여 경량화된 분류 헤드를 구성하므로, 분류 연산 비용을 최소화하면서도 성능 저하를 방지한다. 마지막 basic block에서는 최종 컨볼루션 계층에 대해 고유 랭크(intrinsic rank)를 산출하여 이 값을 근거로 블록 내부 구조를 동적으로 최적화하는 랭크 기반 블록 설계(rank-guided block desing)를 도입했다. CIB 구조를 채택하면서 공간적 특징 추출 때는 Depth-wise Convolution을, 채널 간 정보 처리를 할 때는 Point-wise Convolution을 전담하도록 분리함으로써 불필요한 연산을 줄이고, 추론 속도와 정확도 모두에서 개선을 달성했다[24].



<그림 7> YOLO-v10 model architecture with CIB integration [25]

<그림 8>은 YOLOv10이 후처리 단계의 중복 예측 문제를 어떻게 해결했는지를 나타낸다. 기존 모델은 one-to-many 레이블 할당(one ground truth ↔ 다수의 positive sample) 전략을 사용했었다. 이는 학습 성능은 높아지지만, 추론 시에는 반드시 NMS가 필요하여 추론 속도가 느려지고, NMS의 하이퍼파라미터에 민감해져 완전한 end-to-end 배포를 저해한다[26]. YOLOv10 에서는 Dual Label Assignments 구조를 도입하여 학습 시에는 one-to-many Head로 풍부한 지도 신호를 확보하고, 추론 시에는 one-to-one Head만을 사용해 NMS 없이도 중복 예측을 억제한다. 이때, 다음에 제시된 Consistent Matching Metric을 활용하여 예측과 인스턴스의 bbox(bhat, b)와 각 예측 박스의 신뢰도(s), 분류 점수(p), 겹침 정도(IoU)를 일관되게 평가함으로써 중복 제거 성능을 더욱 향상시켰다[24].

$$m(\alpha,\beta) = s \cdot p^{\alpha} \cdot IoU(\hat{b},b)^{\beta}$$



<그림 8> Consistent dual assignments for NMS-free training [24]

3.2. YOLOv10의 실시간 검출 파이프라인

본 절에서는 Wang et al.(2024)[24]에서 제안한 YOLOv10의 실시간 검출 파이프라인을 중심으로 입력 전처리부터 추론 최적화까지의 전 과정을 요약하여 기술한다.

1) 입력 전처리

입력 이미지는 YOLOv8[26]의 전처리 방식을 그대로 사용하므로 640x640으로 리사이즈하고, 픽셀 값을 [0,1] 범위로 정규화한다.

2) 순전파(Inference Flow)

<그림 8>에서 볼 수 있듯, 추론(Inference)하는 동안에는 one-to-many Head는 버리고 one-to-one Head만 사용하여 예측하게 되는데, 이로써 NMS 없이도 end-to-end 파이프라인이 유지되며 추가적인 추론 오버헤드가 발생하지 않게 된다.

3) 후처리(Post-processing)

순전파를 통해 생성된 모든 예측 박스는 NMS 없이 일관된 매칭 지표를 이용하여 디코딩된다. 구체적으로, one-to-one Head는 각 ground-truth마다 top-1 선택 전략을 적용하여 매칭 점수가 가장 높은 하나의 예측만을 최종 출력으로 채택한다. 이 방식은 Hungarian 매칭[28]과 동등한 검출 성능을 유지하면서도 매칭 연산에 드는 계산량과 학습·추론 시간을 크게 절감하게 한다.

4) 추론 최적화 및 성능

원논문에서 end-to-end 속도 벤치마크는 문헌[26] 방식을 따랐다. NMS의 실행 시간은 입력 영상에 영향을 받으므로, COCO val set에서 배치 크기(batch size)를 1로 설정하여 지연시간을 측정했다. 검증에 사용된 것과 동일한 NMS 하이퍼파라미터를 적용했으며, TensorRT efficientNMSPlugin을 후처리 단계에 추가하고 I/O 오버헤드는 제외한 뒤 모든 이미지에 걸쳐 평균 latency를 산출했다.

YOLOv8 대비 YOLOv10의 주요 개선 지표를 모델 크기별로 정리한 결과는 <표 1>과 같다. 이로써 YOLOv10은 모든 스케일에서 연산량과 지연 시간을 획기적으로 줄이면서도 정확도를 더욱 끌어올려, 실시간 검출에 최적화된 trade-off를 달성했음을 확인할 수 있다.

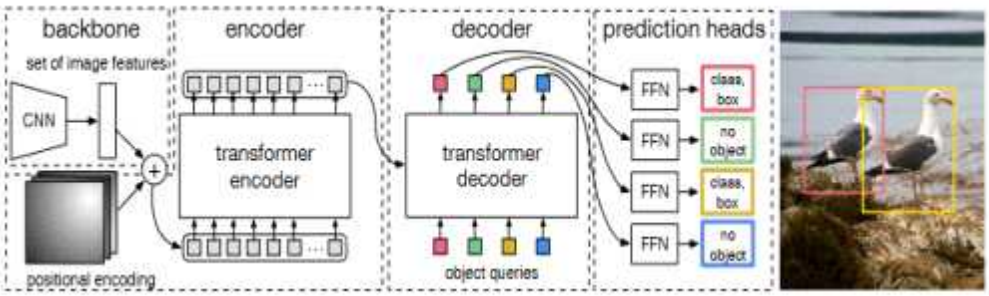
<표 1> YOLOv10과 YOLOv8 성능 비교

규모	△AP (YOLOv10-YOLOv8)	Params ↓(%)	FLOPs ↓ (%)	Latency ↓ (%)
N	+1.2	28	23	70
S	+1.4	36	24	65
M	+0.5	41	25	50
L	+0.3	44	27	41
X	+0.5	57	38	37

3.3. RT-DETR 아키텍처 및 특징

RT-DETR(Real-Time DEtection TRansformer)은 DETR의 end-to-end 학습 장점을 유지하면서도, Transformer encoder-decoder 구조의 불필요한 연산 중복을 제거하여 실시간 검출 요구 사항을 만족하도록 설계된 모델이다.

DETR(DEtection TRansformer)은 표준 CNN 및 Transformer encoder-decorder를 결합하고, 예측된 객체와 ground-truth 간 이분 매칭(bipartite matching) 기반의 집합 손실(set loss)을 사용해 RPN, 앵커 박스, NMS 없이 end-to-end로 학습된다[28]. <그림 9>는 DETR의 전체 파이프라인을 요약한다. DETR은 RPN, 앵커 박스, NMS 없이도 end-to-end 학습이 가능한 구조적 장점을 지니고 있으나 학습 수렴 속도가 느리고 연산 비용이 높아, 실시간 검출에 필요한 지연 시간 기준을 충족하지 못한다.



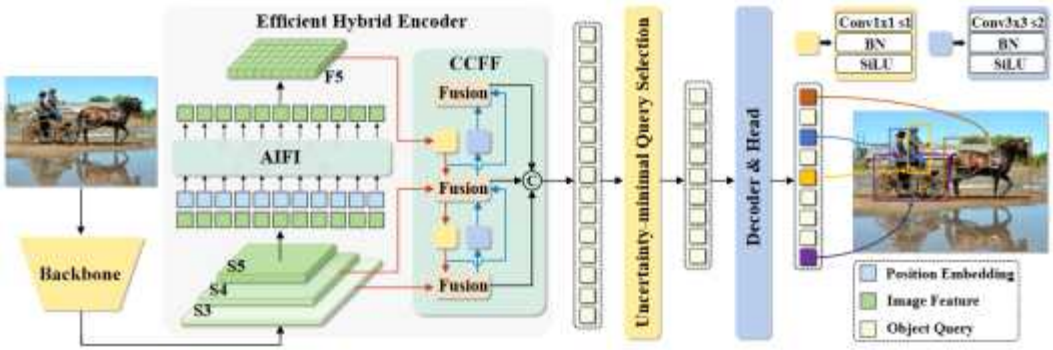
<그림 9> DETR 모델 아키텍처 [28]

또한 NMS 임계값을 별도로 설정해야 하므로 실시간 배포 과정이 복잡해지는 단점이 있다[26]. 특히 SMT 공정에서와 같이 고해상도 이미지에 다수의 작은 결함 객체가 분포하는 경우, 연산량이 급격히 증가하며 수렴 속도 저하

가 더욱 두드러진다. 또한 단일 스케일 특징만을 사용하는 구조는 다양한 크기의 미세 결함을 효과적으로 검출하기에 한계가 있으며, 이는 실제 산업 응용에서 실용성을 저해하는 요인이 될 수 있다.

RT-DETR은 DETR 모델의 구조적 한계인 불필요한 계산 중복 문제를 줄이고 정확도 및 실시간 성능 개선을 목표로 연구되었다.

기존 DETR에서는 학습 수렴 속도를 높이기 위해 다중 스케일 특징을 도입했지만[29], 이는 인코더에 공급되는 시퀀스 길이를 크게 증가시켜 오히려 연산 부담을 가중시키는 문제가 있었다. 이를 해결하기 위해 RT-DETR에서는 인코더 구조를 재설계하였으며 전체 아키텍처는 <그림 10>과 같다[26].



<그림 10> RT-DETR 모델 아키텍처 [26]

모델은 백본(Backbone), 효율적 하이브리드 인코더(Hybrid Encoder), Auxiliary Prediction Head를 포함한 트랜스포머 디코더(Transformer decoder)로 구성된다. 백본에서 추출된 F3~F5 스케일의 피처 맵은 하이브리드 인코더를 거쳐 정제 및 융합된 뒤, IoU 기반의 uncertainty-minimal query selection

을 통해 디코더로 전달되어 최종 객체 예측을 수행한다.

효율적 하이브리드 인코더는 다음 두 단계로 구성된다.

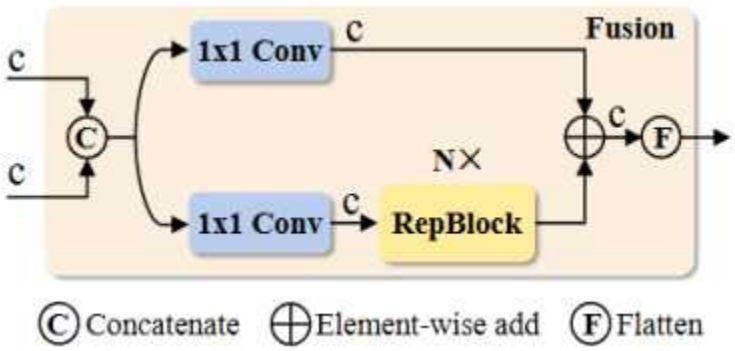
(1) Attention-based Intra-scale Feature Interaction (AIFI)

고수준 피처맵(S5)에서만 self-attention을 적용해 시맨틱 상호작용을 강화하고, 연산 비용을 최소화한다.

(2) CNN-based Cross-scale Feature Fusion (CCFF)

1×1 Conv로 채널을 정렬한 뒤 여러 개의 RepConv 블록을 거쳐 인접 스케일 피처를 단계적으로 결합한다.

각 Fusion Block은 "1×1 Conv → N×RepConv → 1×1 Conv" 구조이며, <그림 11>에 상세히 나타내었다. 이처럼 AIFI와 CCFF를 통해 생성된 S3-S4-S5 멀티스케일 피처를 토큰화(Tokenization)하여 Transformer 인코더에 입력함으로써, 연산 효율과 표현력을 동시에 확보할 수 있다[26].



<그림 11> CCFF의 Fusion Block 구조 [26]

이러한 구조적 설계는 DETR의 핵심 한계였던 단일 스케일 피처 사용, 과도한 self-attention 연산, 쿼리 선택의 비효율성을 각각 다음과 같이 개선하였다.

첫째, 피처 맵은 기존의 단일 해상도 대신 F3~F5의 다중 스케일에서 추출되어, 작은 결함 객체도 효과적으로 표현할 수 있다.

둘째, self-attention 연산을 S5에만 selective하게 적용(AIFI)함으로써 표현력은 유지하면서도 연산량은 크게 절감된다.

셋째, 디코더에는 고정된 쿼리 대신 IoU 예측 기반으로 선택된 상위 토큰만을 전달하여, 더욱 신뢰도 높은 예측을 도출한다.

이와 같은 구조적 개선은 기존 DETR 대비 실시간 적용 가능성을 실질적으로 향상시킬 뿐 아니라, 산업 현장과 같은 고정밀 요구 환경에서도 성능 저하 없이 활용될 수 있도록 한다.

한편, RT-DETR은 기존의 YOLO 계열이나 Faster R-CNN처럼 anchor 설정이 필요한 객체 검출 모델들과 달리, anchor-free 방식으로 객체 위치를 직접 예측하는 구조를 갖는다[26]. 기존 anchor-based 모델들은 데이터셋에 따라 anchor의 크기나 비율 설정이 성능에 큰 영향을 미치며, 최적화되지 않을 경우 작은 객체나 드문 클래스 검출에 불리하다. 반면, RT-DETR은 사전 정의된 anchor 없이 객체를 예측함으로써 다양한 크기와 형태의 결함이 혼재된 SMT 납땜 패드 영역에서도 보다 안정적인 검출 성능을 보인다. 이로 인해, 생산 현장에서 반복되지 않는 새로운 불량 유형에 대해서도 빠르게 대응할 수 있는 유연성을 확보할 수 있다.

3.4. RT-DETR 기반 실시간 객체 검출 워크플로우

본 목차는 RT-DETR 논문[26]에서 제안된 end-to-end 추론 파이프라인을 단계별로 정리한다. <그림 12>는 전체 워크플로우를 시각화한 도식이다.

(1) 입력 전처리

RT-DETR 및 YOLO 모델(N/S/M/L/X)은 640×640 해상도를 사용하며, 그 외 DETR 계열은 800×1,333 해상도를 사용한다. 필요 시 zero-padding을 추가하고 ImageNet 평균·표준편차로 정규화한 뒤 Inference 벤치마크용 배치 크기 1로 모델에 전달한다.

(2) Backbone

ResNet 계열 백본에서 S3, S4, S5 세 가지 해상도의 피처 맵을 추출한다.

(3) Efficient Hybrid Encoder

AIFI: S5 피처맵에만 self-attention을 적용해 시맨틱 상호작용을 강화한다.

CCFF: S3 - S5 피처맵을 1×1 Conv와 RepConv 블록으로 단계별 융합하여 멀티스케일 표현력을 확보한다.

(4) IoU-Aware Query Selection

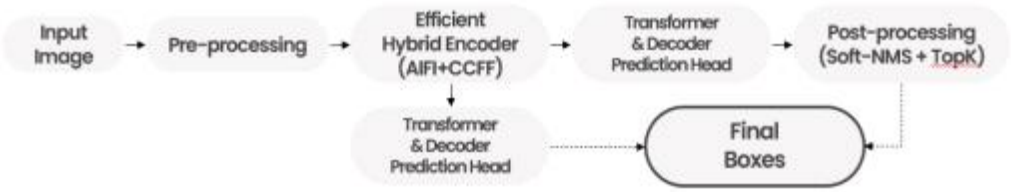
인코더 출력 토큰마다 IoU 예측을 수행하고, 예측 점수가 높은 상위 K개의 토큰만을 디코더 입력으로 선택한다.

(5) Transformer Decoder & Prediction Head

선택된 쿼리들이 multi-head self-attention 및 cross-attention을 거쳐 정제된 뒤, classification과 bounding-box regression을 통해 raw 예측 결과를 생성한다.

(6) 후처리 (선택적)

confidence thresholding을 적용하고, 필요 시 Soft-NMS 또는 Top-K selection을 통해 최종 바운딩 박스만을 출력한다.



<그림 12> RT-DETR End-to-End 추론 워크플로우

위에서 살펴본 바와 같이, YOLOv10과 RT-DETR은 모두 end-to-end 객체 검출 파이프라인을 구현하면서도 실시간 응용에 필요한 속도-정확도 트레이드오프를 탁월하게 달성한 모델이다. 따라서 본 연구에서는 두 모델을 PCB 솔더 페이스트 도포 결함 검출 실험에 적용하여, 각 모델의 실제 성능과 장단점을 비교 및 분석하고자 한다.

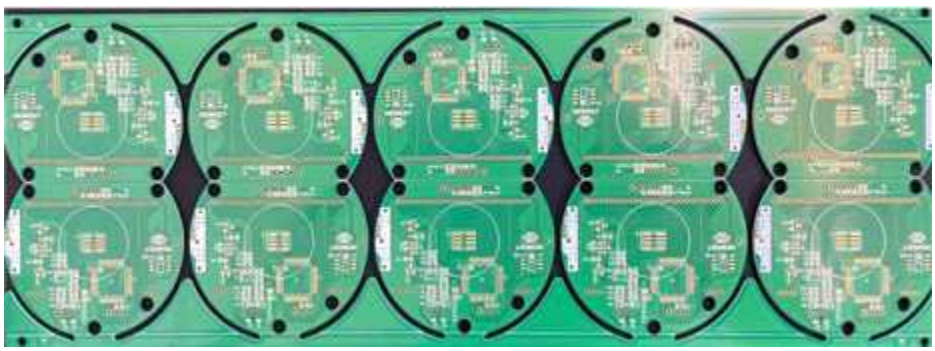
Ⅳ. YOLO 및 RT-DETR를 이용한 PCB 솔더 페이스트 도포 결함 검출 구현

4.1. 데이터 구축 및 전처리

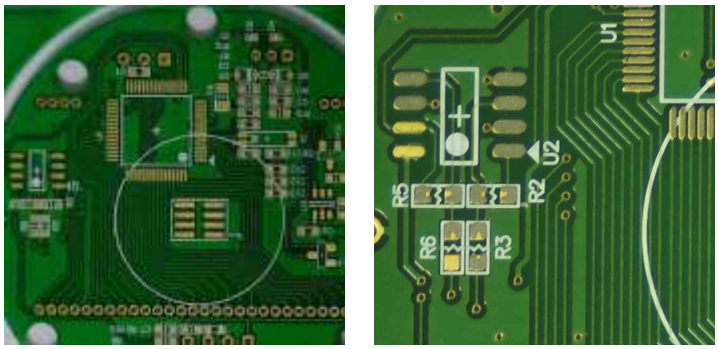
본 논문에서 사용한 시료는 스크린 프린팅 공정을 거친 직후 솔더 페이스트가 일부 도포되지 않은 불량 PCB 90장과 솔더 페이스트가 전혀 도포되지 않은 Bare PCB 70장으로 SMT 제조업체로부터 직접 제공받았다.

실제 생산라인에서 불량 데이터를 충분히 확보하는 데에는 여러 현실적인 제약이 따른다. 이에 따라 본 연구에서는 위 두 종류의 PCB를 모두 불량 데이터로 간주하여 학습 데이터셋을 구성하였다.

실험에 사용된 영상은 Sony IMX 298(16 MP) 카메라로 촬영하였으며, 고해상도 이미지를 기반으로 정밀한 불량 검출을 가능하게 하였다.



<그림 13> 실험에 사용된 PCB 이미지



(a) (b)

<그림 14> 고해상도 PCB 이미지의 분할 및 학습용 이미지 재구성 예시
(a)분할된 납땜 패드 영역
(b)640×640 크기로 재분할된 학습용 이미지

실험에 사용된 PCB는 <그림 13>과 같으며 전체를 촬영했을 때 납땜 패드가 매우 작아 인식이 어려우므로 확대하여 촬영하였다. 하나의 PCB에는 <그림 14>의 (a)와 같이 동일한 형태의 납땜 패드가 10개 포함되어 있으며, 이들 각각을 분할하여 촬영하였다. 4032×3040 고해상도 원본 이미지에서 객체를 더욱 정밀하게 보기 위해서 <그림 14>의 (b)와 같이 640×640 크기로 재분할하여 학습 데이터셋을 구성하였다.

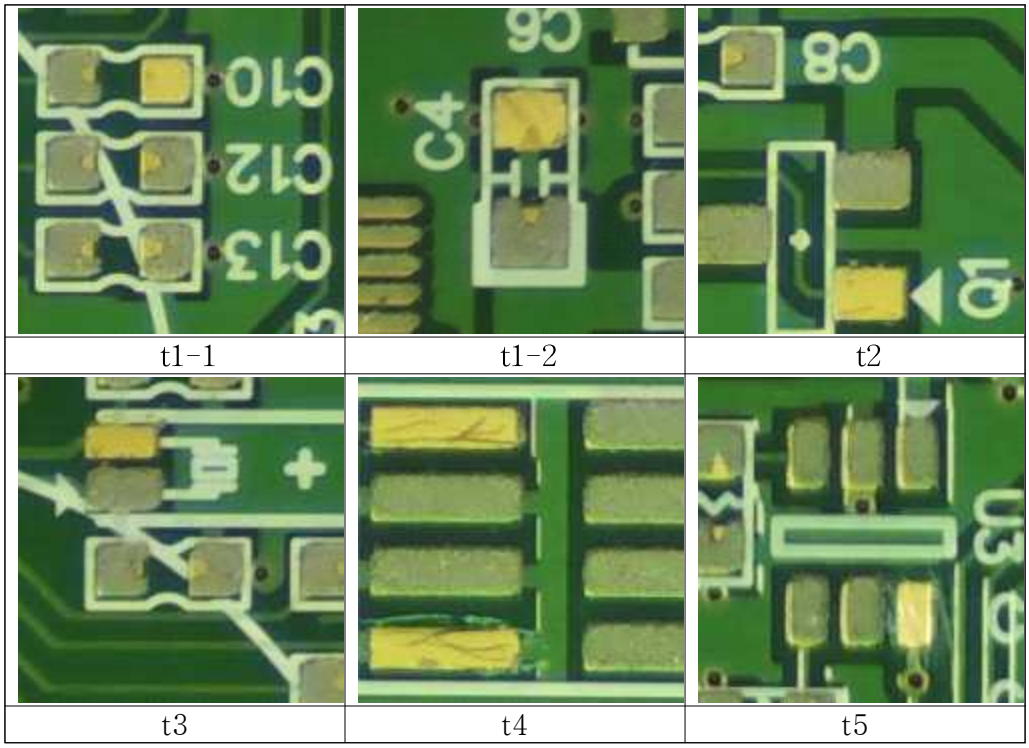
전체 6,724장의 학습용 이미지는 <표 2>와 같이 Train, Validation, Test를 6 : 2 : 2 비율로 분할하였고, 학습 데이터는 <그림 15>와 같이 불량 유형에 따라 임의 불량인 Bare PCB 3,060장과 솔더 페이스트가 일부 미도포된 불량(Solder paste missing) PCB 3,664장으로 구성되어 있다.

<그림 15> 학습 데이터셋의 불량 유형별 구성

<표 2> PCB 솔더 페이스트 이미지 데이터셋 구성

구분	Train	Validation	Test
Count	4,034	1,345	1,345

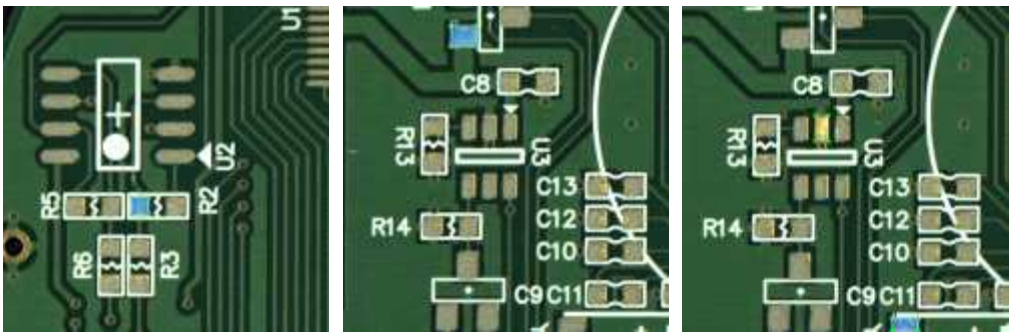
솔더 페이스트가 일부 미도포된 불량 유형은 <그림 16>과 같다. 각 유형은 납땜 패드의 모양과 크기가 비슷하면 동일한 클래스로 보고 t1부터 t5로 라벨링했다. 사진에서 납땜 패드에 노란색이 보이면 솔더 페이스트가 미도포된 상태인 것으로 불량 데이터이며 하나의 이미지에는 최소 1개에서 최대 9개의 불량이 포함되어 있다.

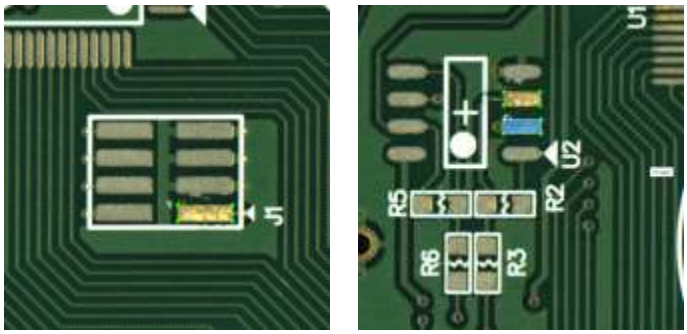


t1-1	t1-2	t2
t3	t4	t5

<그림 16> 솔더 페이스트 미도포 불량 유형

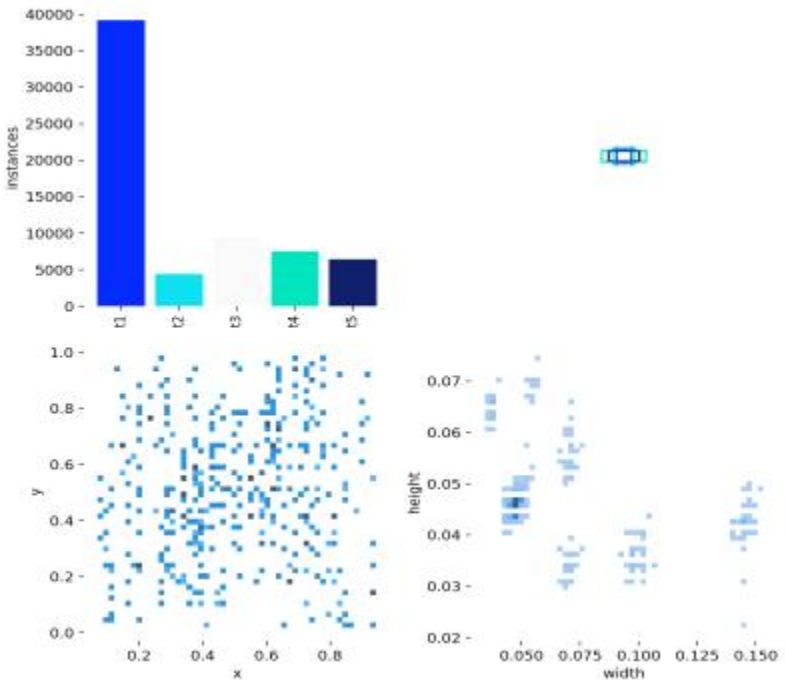
각 객체에 대한 클래스 라벨링은 Python 기반 이미지 주석 도구인 labelImg를 사용하여 직사각형 형태로 박스를 그렸으며, 예시는 <그림 17>과 같다.





<그림 17> 불량 데이터 어노테이션 예시

본 실험을 진행하기에 앞서 YOLOv10-N 모델에서 테스트 실험을 진행했을 때, 데이터셋의 분포는 <그림 18>과 같았다. t1 클래스의 수가 다른 클래스의 수보다 많아 클래스 불균형이 심한 것을 확인할 수 있었다. 따라서, t1부터 t5까지 각각 0.5, 1.5, 1.5, 1.5, 1.5로 클래스별 가중치를 두어 실험을 진행하였다.

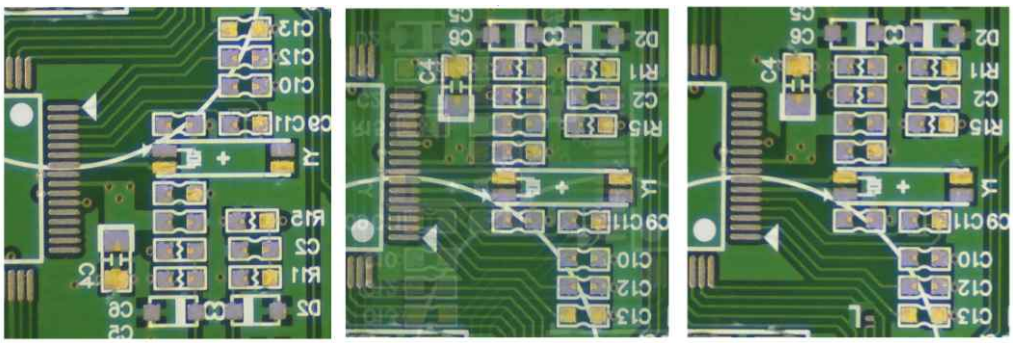


<그림 18> 학습 데이터 분포 분석

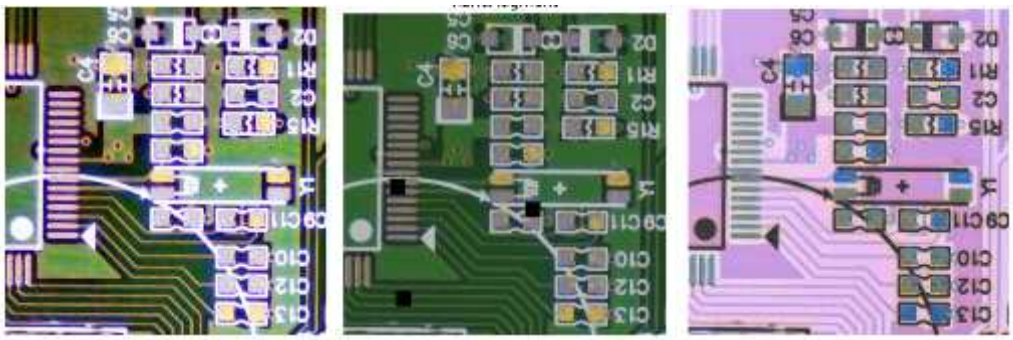
4.2. 데이터 증강

본 논문에서는 첫 번째로 동일한 하이퍼파라미터에서 여러 규모의 YOLOv10과 RT-DETR-L 간의 성능을 먼저 비교하였다. mAP@50 값이 RT-DETR-L 모델이 가장 높게 나왔으므로 해당 모델을 기준으로 데이터 증강을 통해 성능 개선을 진행하는 두번째 실험을 진행하였다.

RT-DETR은 복잡한 데이터 증강 기법을 최소화하여 학습 효율성과 추론 속도를 높이는 것을 목적으로 한다. [30]의 연구에서 동적 데이터 증강(Dynamic Data Augmentation)을 진행하는데 이는 학습 초기에 검출기의 일반화 성능이 낮다는 점을 고려해 강한 데이터 증강을 적용하고, 학습 후반부에서는 실제 테스트 데이터에 적응하기 위해 증강 강도를 줄였다고 명시되어 있다. [31] 연구에서는 RT-DETR 기반 모델에서 Mosaic 증강 기법을 사용하지 않고도 높은 정확도를 달성한 사례를 보여준다. 따라서, 본 논문에서도 Mosaic 기법을 사용하지 않고 flipud, mixup, copy_paste, auto_augment 파라미터만을 조정하여 학습을 진행했다<그림 19>. ultralytics 라이브러리를 사용했을 때, default 설정과 비교를 위한 데이터 증강 관련 하이퍼파라미터 설정은 <표 3>과 같다.



(a) (b) (c)



(d) (e) (f)

<그림 19> 데이터 증강 예시
(a) flipud=0.5 (b) mixup=0.2 (c) copy_paste=0.2
(d)AutoAugment (e)RandAugment (f)AugMix

<표 3> RT-DETR의 데이터 증강 관련 하이퍼파라미터

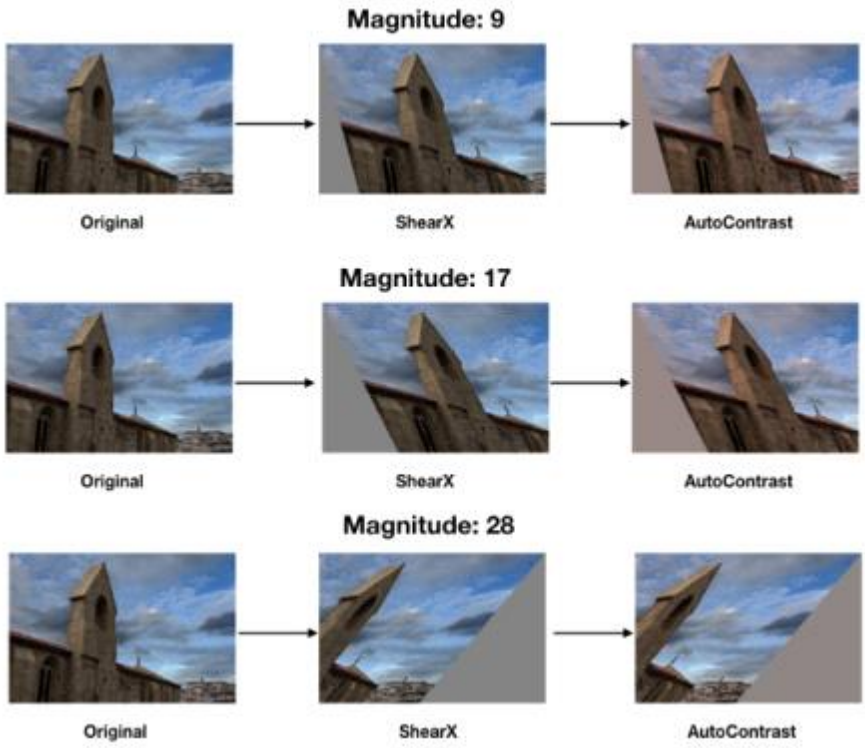
	Default	Custom		
		실험 1	실험 2	실험 3
hsv_h	0.015	0	0	0
hsv_s	0.7	0.7	0.7	0.7
hsv_v	0.4	0.4	0.4	0.4
flipud	0	0.5	0.5	0.5
mixup	0	0.2	0.2	0.2
copy_paste	0	0.2	0.2	0.2
auto_augment	autoaugment	autoaugment	randaugment	augmix

hsv_h, s, v는 각각 임의의 색조 증강, 채도 증강, 밝기 조절 담당하며 flipud은 수직 이미지 뒤집기 확률이다. mixup은 두 이미지를 혼합하는 믹스업 증강을 사용할 확률, copy_paste는 복사-붙여넣기 증강을 사용할 확률이다. 모든 하이퍼파라미터는 0.0~1.0 범위에서 확률적으로 작동한다.

ultralytics 라이브러리에서 제공하는 자동 증강(auto_augment)의 옵션으로는 AutoAugment[32], RandAugment[33], AugMix[34], None이 있다.

autoaugment는 주어진 데이터셋에 대해 검증 정확도를 최적화할 수 있는 이미지 변환 조합(policy)을 강화 학습(Reinforcement Learning, RL) 기반으로 학습하는 자동화된 데이터 증강 기법이다. 이때 사용 가능한 이미지 변환 후보에는 기하학적 변환(shear, translation, rotation), 화소 조작(contrast, brightness, color, sharpness), 색상 반전 및 포스터화(solarization, posterization, invert), 히스토그램 평활화, Cutout, Sample Pairing 등이 포함된다.

AutoAugment의 한계를 보완해 증강 정책을 따로 학습시키지 않고도 유사한 성능을 달성할 수 있도록 고안된 데이터 증강 방식이 RandAugment이다. RandAugment는 각 이미지에 적용할 증강 기법을 사전에 정의된 목록에서 균등 확률로 무작위 선택하는데, 하나의 이미지에 적용할 증강 기법의 개수(N)와 각 기법의 강도(M)인 두 가지 하이퍼파라미터만으로 설정된다. 변환 목록에는 Identity, AutoContrast, Equalize, Rotate, Solarize, Color, Posterize, Brightness, Sharpness, ShearX/Y, TranslateX/Y 등이 포함되며 증강 예시 이미지는 <그림 20>과 같다.



<그림 20> RandAugment 증강 예시 이미지 [33]

RandAugment는 증강 정책 학습 단계를 생략하면서 다양한 비전 과제에서 강건한 성능을 보여주는 것으로 보고되었으며, 이와 관련된 간단한 비교는 기술 블로그 형태의 웹 매거진에서도 소개된 바 있다[35].

AugMix는 도메인 변화나 데이터 분포 변화에 대한 강건성(robustness)을 향상시키기 위해 설계된 데이터 증강 기법이다. AutoAugment와 유사한 이미지 변환 기법을 기반으로 하되, contrast, brightness, color, sharpness, Cutout 등 일부 기법은 제거되었으며 배포 환경에서 예기치 않은 데이터 변화에 대한 모델의 강건성을 향상시키는 데 효과적인 것으로 소개되고 있다[35].

4.3. 학습 환경 및 설정

본 연구는 <표 4>와 같은 하드웨어 및 소프트웨어 환경에서 수행되었으며, 사용된 모든 소프트웨어는 Windows 10 Pro for Workstations 운영 체제 기반에서 구동되었다. GPU(NVIDIA GeForce RTX 3080) 및 128GB의 메모리를 기반으로 대용량 이미지 데이터를 처리하고, PyTorch 프레임워크를 이용하여 YOLO 및 RT-DETR 모델을 학습하였다. 학습은 Jupyter Notebook 환경에서 진행했으며, 객체 검출을 위한 어노테이션 작업에는 LabelImg를 사용하였다.

<표 4> 실험 환경 구성

구분	구성요소
OS	Windows 10 Pro for Workstations
HW	CPU: Intel(R) Xenon(R) W-2223 CPU @ 3.60Hz GPU: NVIDIA GeForce RTX 3080 RAM: 128 GB
SW	Python 3.9 PyTorch 1.31.1 +cu117 Jupyter 1.0.0 LabelImg

YOLOv10과 RT-DETR-L 모델의 주요 학습 설정은 <표 5>에 요약하였다. 하이퍼파라미터는 대부분 동일한 조건에서 비교 수행하였으며, optimizer와 momentum, 초기 학습률(lr0)은 각각의 모델 특성에 따라 Ultralytics 프레임워크에 의해 자동으로 설정된 값이다.

<표 5> YOLOv10과 RT-DETR-L의 하이퍼파라미터

항목	설정값
epoch	20
batch_size	16
image_size	640
Optimizer	AdamW
lr0(초기 학습률)	0.00111
Momentum	0.9
Dropout	0.5

4.4. YOLO 기반 결함 검출 모델 구현

본 연구에서는 Ultralytics에서 제공하는 YOLOv10 모델을 기반으로 결함 검출 실험을 수행하였다. Miniconda를 활용해 가상 환경을 구성한 후, 공식 YOLO 리포지터리를 통해 학습 환경을 구축하였다. YOLOv10은 모델의 경량화 수준에 따라 n, s, m, l, x의 다섯 가지 변형된 모델을 지원하며, 본 연구에서는 모든 모델에서 학습하고 성능을 비교 분석하였다.

모델 구조는 Ultralytics에서 제공하는 기본 아키텍처를 변경하지 않고 활용하였으며, 하이퍼파라미터는 학습 데이터의 특성에 따라 주요 항목인 Epoch, Image Size, Dropout만 조정하였다. YOLOv10의 상세 계층 구조는 <그림 21>에, 모델의 네트워크 구조 요약은 <표 6>에 제시하였다. 모델별 성능 비교 결과는 V장의 실험 결과 및 고찰 절에서 상세히 다루었다.

	from	n	params	module	arguments
0	-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
1	-1	1	4672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
2	-1	1	7360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
3	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
4	-1	2	49664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
5	-1	1	9856	ultralytics.nn.modules.block.SCDown	[64, 128, 3, 2]
6	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
7	-1	1	36096	ultralytics.nn.modules.block.SCDown	[128, 256, 3, 2]
8	-1	1	460288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
9	-1	1	164608	ultralytics.nn.modules.block.SPPF	[256, 256, 5]
10	-1	1	249728	ultralytics.nn.modules.block.PSA	[256, 256]
11	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
12	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
13	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
14	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
15	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
16	-1	1	37248	ultralytics.nn.modules.block.C2f	[192, 64, 1]
17	-1	1	36992	ultralytics.nn.modules.conv.Conv	[64, 64, 3, 2]
18	[-1, 13]	1	0	ultralytics.nn.modules.conv.Concat	[1]
19	-1	1	123648	ultralytics.nn.modules.block.C2f	[192, 128, 1]
20	-1	1	18048	ultralytics.nn.modules.block.SCDown	[128, 128, 3, 2]
21	[-1, 10]	1	0	ultralytics.nn.modules.conv.Concat	[1]
22	-1	1	282624	ultralytics.nn.modules.block.C2fCIB	[384, 256, 1, True, True]
23	[16, 19, 22]	1	863278	ultralytics.nn.modules.head.v10Detect	[5, [64, 128, 256]]

<그림 21> YOLOv10-N 네트워크 계층 구조

<표 6> YOLOv10 각 버전의 네트워크 요약 비교

모델	계층 수(Layers)	파라미터 수 (Parameters)	연산량 (GFLOPs)
YOLOv10-N	223	2,708,990	8.4
YOLOv10-S	234	8,070,222	24.8
YOLOv10-M	288	16,489,918	64.0
YOLOv10-L	364	25,773,038	127.2
YOLOv10-X	400	31,664,510	171.0

4.5. RT-DETR 기반 결함 검출 모델 구현

Ultralytics에서 제공하는 RT-DETR ~ YOLOv10과 동일한 환경에서 실험을 진행하였으며, 지원하는 모델 유형 중 RT-DETR-L(Large) 버전을 활용하였다. 아키텍처 역시 변경하지 않고 그대로 사용하였다. 하이퍼파라미터는 YOLOv10과의 비교 실험을 위해 동일한 기준에서 설정하되, 일부 설정(optimizer, 학습률 등)은 자동으로 결정되도록 구성하였다.

RT-DETR의 네트워크 계층 구조는 <그림 22>에, 요약된 정보는 <표 7>에 제시하였다.

```
                from  n    params  module                                       arguments
  0                -1  1     25248  ultralytics.nn.modules.block.HGStem          [3, 32, 48]
  1                -1  6    155072  ultralytics.nn.modules.block.HGBlock         [48, 48, 128, 3, 6]
  2                -1  1      1408  ultralytics.nn.modules.conv.DWConv           [128, 128, 3, 2, 1, False]
  3                -1  6    839296  ultralytics.nn.modules.block.HGBlock         [128, 96, 512, 3, 6]
  4                -1  1      5632  ultralytics.nn.modules.conv.DWConv           [512, 512, 3, 2, 1, False]
  5                -1  6   1695360  ultralytics.nn.modules.block.HGBlock         [512, 192, 1024, 5, 6, True, False]
  6                -1  6   2055808  ultralytics.nn.modules.block.HGBlock         [1024, 192, 1024, 5, 6, True, True]
  7                -1  6   2055808  ultralytics.nn.modules.block.HGBlock         [1024, 192, 1024, 5, 6, True, True]
  8                -1  1     11264  ultralytics.nn.modules.conv.DWConv           [1024, 1024, 3, 2, 1, False]
  9                -1  6   6708480  ultralytics.nn.modules.block.HGBlock         [1024, 384, 2048, 5, 6, True, False]
 10                -1  1    524800  ultralytics.nn.modules.conv.Conv             [2048, 256, 1, 1, None, 1, 1, False]
 11                -1  1    789760  ultralytics.nn.modules.transformer.AIFI      [256, 1024, 8]
 12                -1  1     66048  ultralytics.nn.modules.conv.Conv             [256, 256, 1, 1]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14                 7  1    262656  ultralytics.nn.modules.conv.Conv             [1024, 256, 1, 1, None, 1, 1, False]
 15          [-2, -1]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 16                -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 17                -1  1     66048  ultralytics.nn.modules.conv.Conv             [256, 256, 1, 1]
 18                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 19                 3  1    131584  ultralytics.nn.modules.conv.Conv             [512, 256, 1, 1, None, 1, 1, False]
 20          [-2, -1]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 22                -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 23          [-1, 17]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 24                -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 25                -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 26          [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 27                -1  3   2232320  ultralytics.nn.modules.block.RepC3           [512, 256, 3]
 28      [21, 24, 27]  1   7312127  ultralytics.nn.modules.head.RTDETRDecoder    [5, [256, 256, 256]]
rt-detr-l summary: 457 layers, 32,816,351 parameters, 32,816,351 gradients, 108.0 GFLOPs
```

<그림 22> RT-DETR-L 네트워크 계층 구조

<표 7> RT-DETR-L 네트워크 요약 정보

항목	값
계층 수(Layers)	457
파라미터 수 (Parameters)	32,816,351
Gradient 수	32,816,351
연산량 (GFLOPs)	108.0

Ⅴ. PCB 솔더 페이스트 도포 결함 검출 실험 결과 및 고찰

5.1. 모델 성능 비교 분석

1) YOLOv10-L 성능 평가

YOLO 모델의 성능 평가는 Precision, Recall, mAP@0.5, 추론시간(FPS)을 기준으로 정량적으로 평가하였으며, 모든 성능 지표는 소수점 네 번째 자리에서 반올림하여 표기하였다.

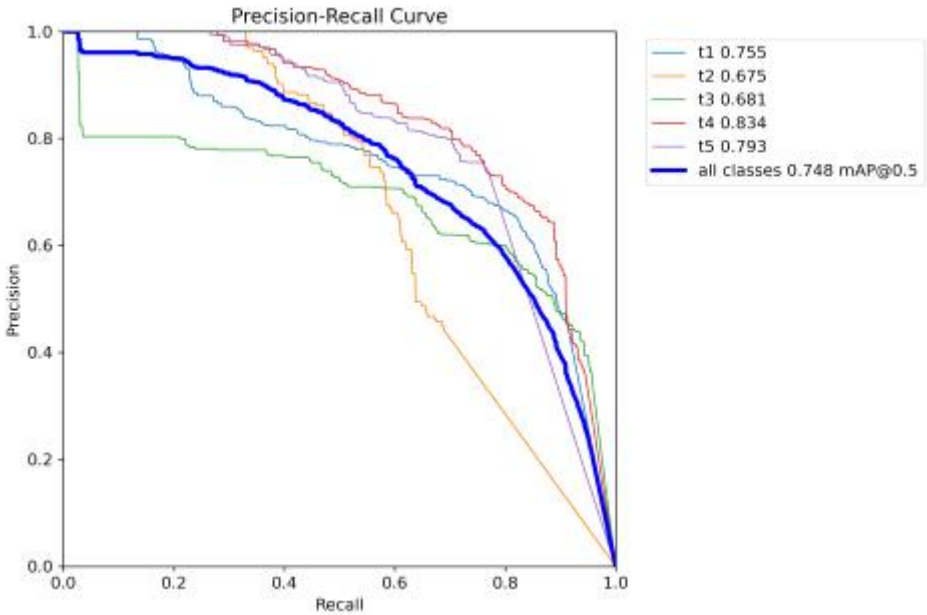
추론 시간은 Ultralytics 프레임워크에서 제공하는 출력 중 preprocess, inference, postprocess 단계의 시간 합산값을 기반으로 산출하였으며, 1장의 이미지를 처리하는 데 소요되는 평균 시간(ms/image)으로 나타냈다.

실험 결과는 <표 8>과 같다. YOLOv10의 다섯 가지 버전 중 YOLOv10-L 모델이 Precision 0.671, Recall 0.699, mAP@0.5 0.748로 가장 우수한 성능을 기록하였고, 약 6시간 45분(6.749 hours)의 학습 시간과 11.4ms의 추론 시간을 보였다. 이에 따라 RT-DETR 모델과의 비교 대상으로 선정하였다.

<표 8> YOLOv10 각 버전에서의 실험 결과

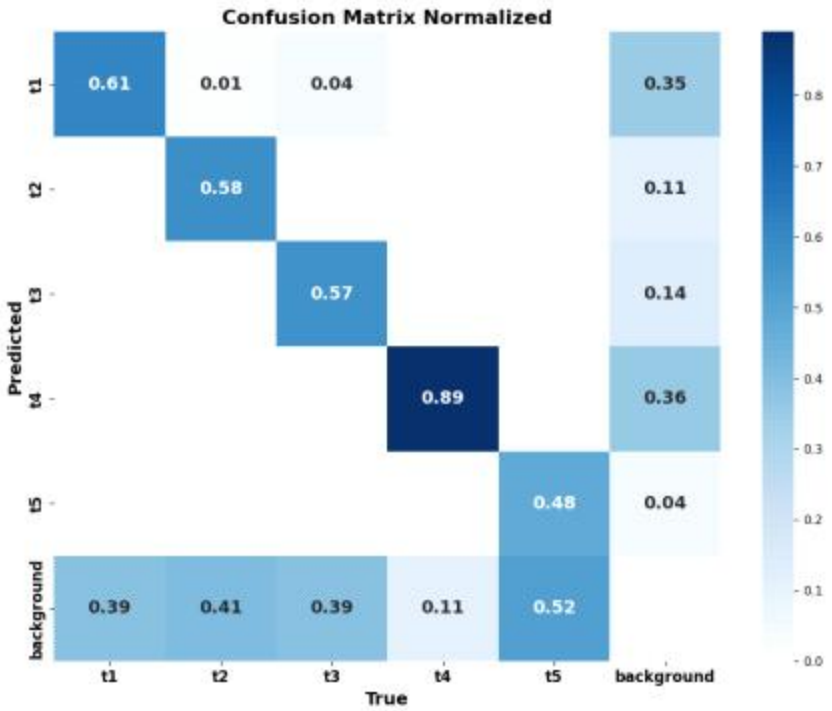
모델	Precision	Recall	mAP@0.5	학습 시간 (시간)	추론 시간 (ms/image)
YOLOv10-N	0.477	0.621	0.499	0.346	2.5
YOLOv10-S	0.637	0.701	0.647	0.386	4
YOLOv10-M	0.605	0.714	0.66	0.922	7.1
YOLOv10-L	0.671	0.699	0.748	6.749	11.4
YOLOv10-X	0.59	0.644	0.615	24.116	16.5

YOLOv10-L 모델의 Precision-Recall(PR) 커브 및 혼동행렬은 <그림 23>과 <그림 24>와 같다. 클래스별 PR 분석 결과, t4(0.834)와 t5(0.793) 클래스에서 높은 Average Precision(AP)을 기록하였다. 반면 t2는 0.675로 상대적으로 낮은 값을 보여 검출 성능 개선이 필요한 클래스로 확인되었다.



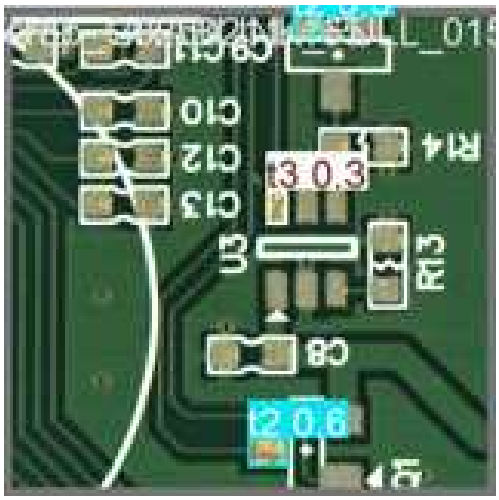
<그림 23> YOLOv10-L Precision-Recall 곡선

혼동행렬은 정규화된 값으로 표현하였으며, 대각선에 가까울수록 정답과 예측이 일치함을 의미한다. t4 클래스는 약 0.89로 비교적 높은 정답 예측 비율을 보였으나, t2(0.58), t3(0.57), t5(0.48) 클래스는 상대적으로 낮은 수치를 보였다. 이는 해당 클래스들이 배경과 시각적 특징이 유사하거나, 다른 클래스와의 경계가 불분명하여 모델이 혼동하기 쉬운 특성을 가지고 있음을 시사한다.



<그림 24> YOLOv10-L Confusion Matrix Normalized

YOLOv10-L 모델의 예측 결과 중 하나를 <그림 25>에 제시하였다. 해당 예시에서는 t2 및 t3 클래스가 배경 위에서 불분명하게 탐지되었으며, confidence score는 각각 0.6, 0.3 수준으로 나타났다. 특히 t3 클래스에서는 매우 낮게 예측되었는데, 이는 혼동행렬에서의 낮은 정답 예측률과 일치한다. 해당 사례는 납땜 패드에서 솔더 페이스트가 도포된 부분과 불량 경계가 모호할 경우, YOLOv10-L 모델이 이를 명확히 구분하기 어려움을 시사한다.



<그림 25> YOLOv10-L의 예측 결과 이미지 (t2, t3 클래스)

2) RT-DETR-L 성능 평가

RT-DETR-L 모델에 대한 성능 분석 결과는 <표 9>와 같으며, Precision은 0.752, Recall은 0.755, mAP@0.5는 0.809로 측정되었다. 학습 시간은 약 8시간 55분(8.916 hours), 추론 시간은 10.0ms/images로 YOLOv10-L 대비 높은 정확도와 실시간의 균형을 달성한 것으로 나타났다.

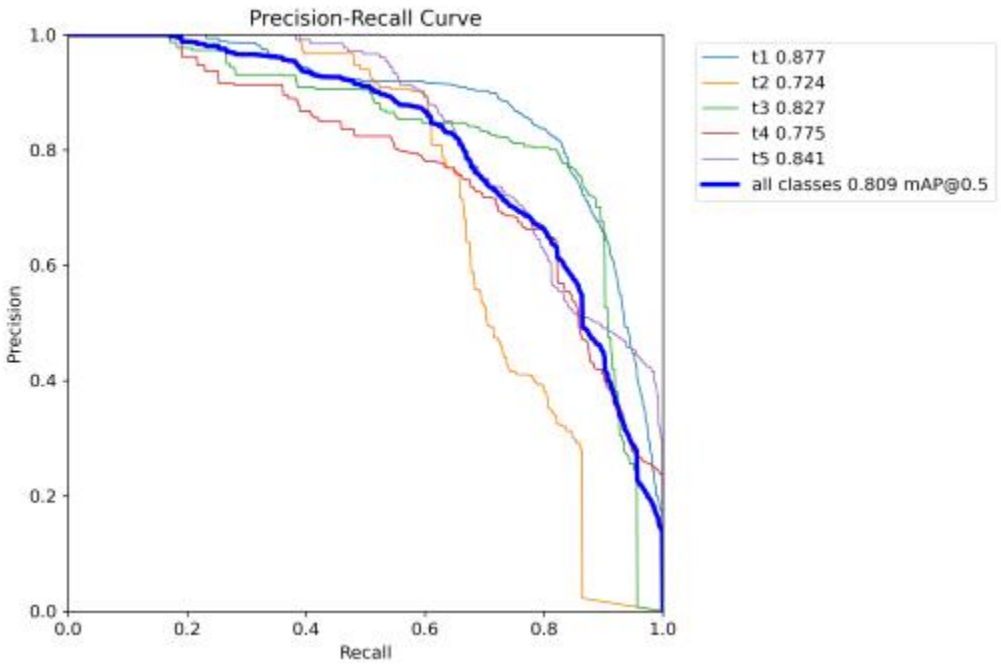
<표 9> RT-DETR-L 실험 결과

모델	Precision	Recall	mAP@0.5	학습시간 (시간)	추론 시간 (ms/image)
RT-DETR-L	0.752	0.755	0.809	8.916	10.0

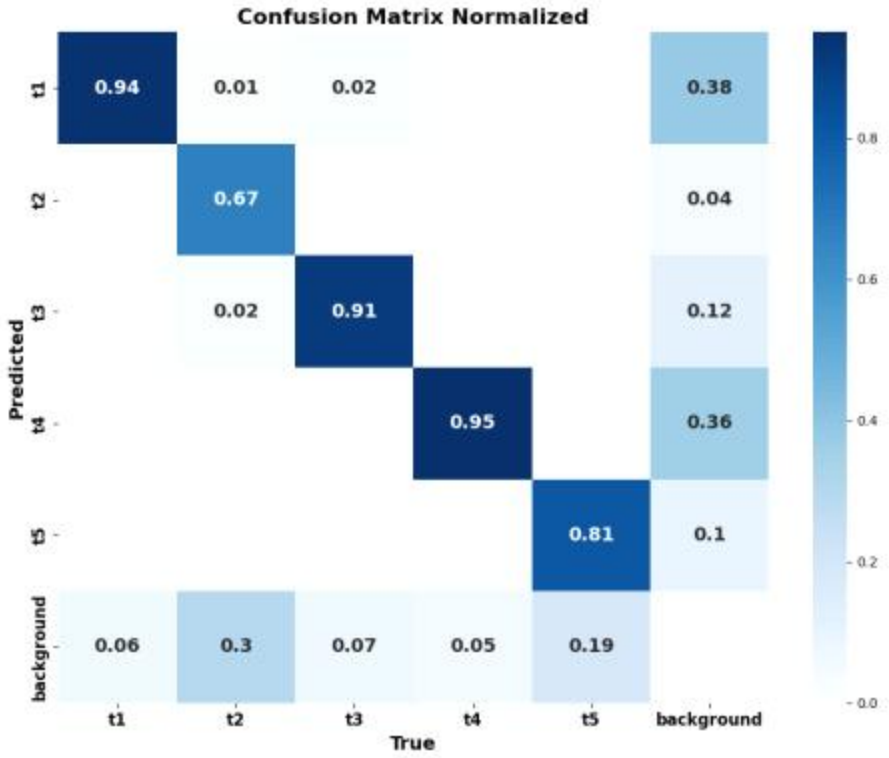
<그림 26>의 RT-DETR 모델의 PR 커브에서 클래스별 AP를 살펴보면 t1 (0.877), t3(0.827), t5(0.841) 클래스에서 높은 성능을 기록하였다. 반면, t2(0.72

4) 클래스는 YOLOv10-L와 마찬가지로 상대적으로 낮은 값을 보여, 배경과의 경계가 모호하거나 시각적 특징이 약한 것으로 판단된다.

혼동행렬 결과는 <그림 27>과 같다. t1, t3, t4 클래스 모두 0.91 이상, t5는 0.81의 정답 예측 비율을 기록하였다. 특히 YOLOv10-L에서는 t5 클래스가 background로 잘못 예측된 비율이 0.52였던 반면, RT-DETR에서는 0.19로크게 감소하였다. 이는 모델이 객체와 배경 간의 경계 구분에 보다 우수한 성능을 보임을 의미한다.



<그림 26> RT-DETR-L Precision-Recall 곡선

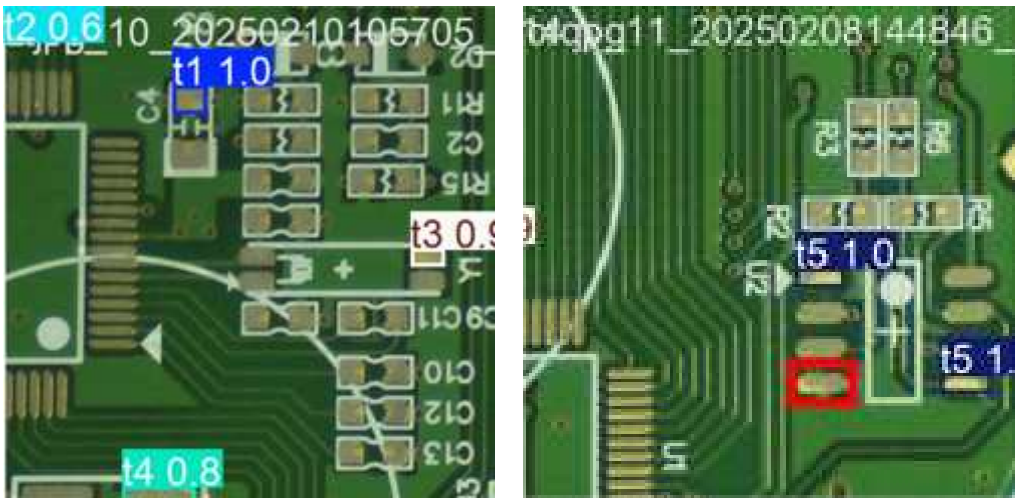


<그림 27> RT-DETR-L Confusion Matrix Normalized

RT-DETR-L 모델의 예측 결과는 <그림 28>과 같다. (a)의 t3 클래스는 YOLOv10-L에서 confidence(0.3)로 탐지되었던 동일한 객체를, RT-DETR-L에서는 0.9 이상의 높은 값으로 정확히 검출하였다. 이는 혼동행렬에서 t3 클래스에 대한 정답 예측률이 RT-DETR-L에서 0.91까지 향상된 결과와도 일치한다.

그러나 RT-DETR-L 모델도 일부 한계를 보이는 경우가 존재한다. <그림 28>의 (b)는 t5 클래스 예측 결과 중 하나로, 동일 클래스 내 다른 인스턴스는 정확히 검출되었으나, 솔더 페이스트가 일부 제거된 불량 영역은 누락(FN)된 사례를 보여준다. 이처럼 RT-DETR-L 모델도 경계가 흐릿하거나 손상이 매

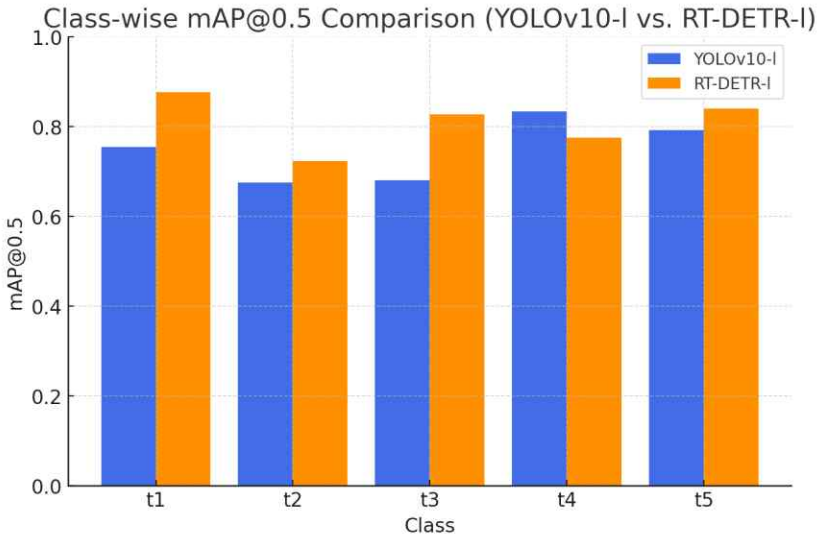
우 경미한 결함에 대해서는 여전히 놓칠 가능성이 있으며, 이는 향후 데이터 다양성 보강 및 미세 결함 인식 능력 향상을 위한 추가 연구가 필요함을 시사한다.



(a) (b)

<그림 28> RT-DETR-L 예측 결과 이미지
(a) t3 클래스 검출 개선 사례 (b) t5 클래스 미탐지 사례

YOLOv10-L과 RT-DETR-L 모델의 성능을 종합적으로 비교한 결과 RT-DETR-L 모델은 전체 mAP@0.5 기준으로 0.809를 기록하여, YOLOv10-L의 0.748 대비 우수한 성능을 보였다. <그림 29>에서 확인할 수 있듯 t1, t3, t5 클래스에서 가장 뚜렷한 성능 차이를 나타내었으며, 이는 PR 커브 및 혼동행렬 분석과도 일치한다.



<그림 29> YOLOv10-L과 RT-DETR-L의 클래스별 mAP@0.5 그래프

또한 YOLOv10-L에서 t5 클래스가 background로 잘못 예측된 비율이 0.52에 달했으나 RT-DETR-L에서는 해당 수치가 0.19로 크게 감소하였으며, 이는 Transformer 기반 인코더-디코더 구조가 객체와 배경의 경계를 보다 효과적으로 학습하고 있음을 보여준다.

RT-DETR-L는 YOLOv10-L 대비 학습 시간을 다소 길지만, 실시간 처리 수준의 추론 시간(10.0ms/image)을 유지하면서도 더 높은 정밀도와 클래스 간 균형 잡힌 검출 성능을 보였다.

이러한 결과를 바탕으로 본 연구는 RT-DETR-L 모델의 성능을 더욱 향상시키기 위해 데이터 증강과 관련된 하이퍼파라미터 최적화 실험을 추가적으로 수행하였다.

5.2. RT-DETR 하이퍼파라미터 최적화

본 실험은 RT-DETR-L 모델의 성능 개선을 위해 데이터 증강 관련 하이퍼파라미터를 조정하는 실험을 수행하였다. 초기 실험에서는 'hsv_h', 'hsv_s', 'hsv_v', 'flipud', 'mixup', 'copy_paste', 'auto_augment' 항목에 대해 임의의 값을 설정한 뒤, 각 요소의 조합이 모델 성능에 미치는 영향을 단계적으로 분석하였다. mAP@0.5를 기준으로 가장 성능이 우수했던 상위 3가지 실험 결과는 <표 10>에 제시하였고, 각각의 설정에 대해 Default 대비 성능 향상 폭을 백분율로 나타낸 비교 결과는 <표 11>에 정리하였다.

<표 10> RT-DETR-L 모델의 데이터 증강 성능 비교

항목	Default	Custom		
		실험 1 (AutoAugment)	실험 2 (RandAugment)	실험 3 (AugMix)
mAP@0.5	0.809	0.96	0.973	0.959

<표 11> mAP@0.5 향상률 비교

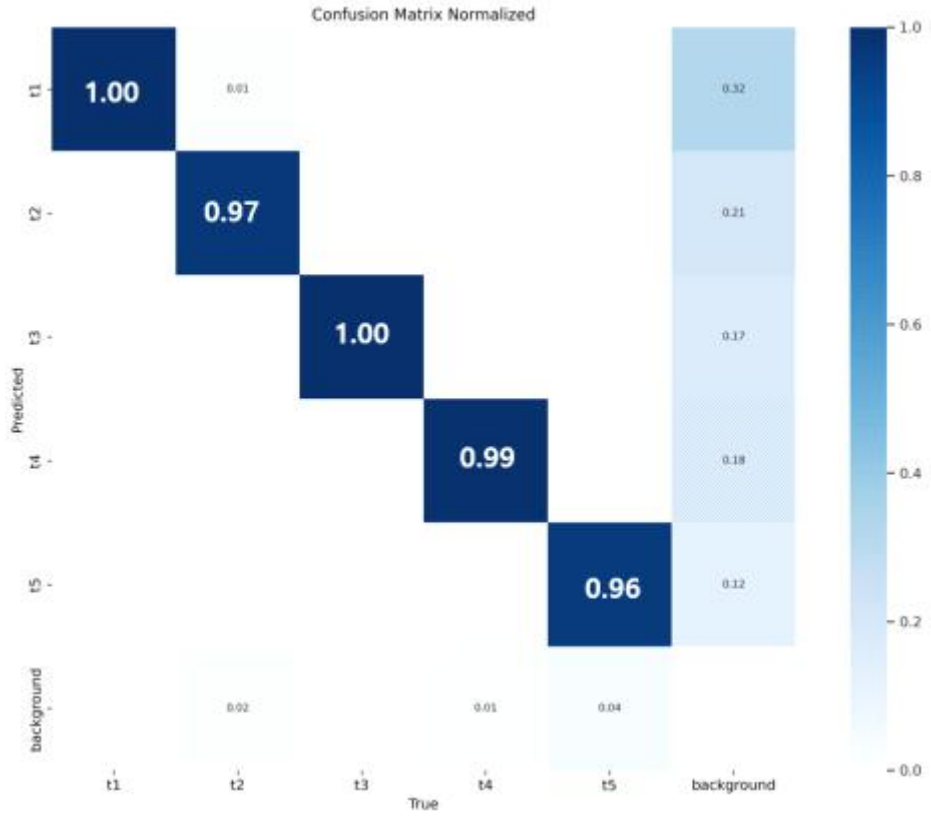
	AutoAugment	RandAugment	AugMix
mAP@0.5 향상률	+15.1%p	+16.4%p	+15.0%p

세 가지 실험 설정 모두에서 Default 대비 mAP@0.5가 유의미하게 향상되었다. 특히 RandAugment를 적용한 실험 2에서는 0.973의 최고 성능을 기록하였으며, 이는 +16.4%p에 해당하는 가장 큰 향상 폭이다.

또한, 기존에 상대적으로 성능이 낮았던 t2, t4 클래스에서 뚜렷한 향상이 관찰되었으며, 실험 2(RandAugment)의 경우 t1~t5 모든 클래스에서 0.9 이상의 정확도를 달성함으로써 클래스 간 성능 편차 또한 감소한 것으로 나타났다.

<그림 30>는 실험 2(RandAugment)의 혼동행렬로, 클래스별 정답 예측 비율이 모두 0.96 이상으로 향상되었음을 보여준다.

예를 들어, Default 설정에서는 t2 클래스가 0.67, t5 클래스가 0.81에 그쳤으나, RandAugment를 적용한 후에는 각각 0.97, 0.96으로 크게 상승하였다. 이러한 결과는 RandAugment 기반 데이터 증강 기법이 시각적 경계가 모호한 결함 클래스의 검출 성능이 실질적으로 개선할 수 있음을 시사한다.



<그림 30> RandAugment 적용 시 혼동행렬 (실험 2)

Ⅵ. 결론

본 논문에서는 인공지능 기반 객체 검출 모델을 활용하여 SMT 공정 중 발생할 수 있는 PCB 기판의 솔더 페이스트 미도포 불량을 검출하는 방법을 제안하였다. 대표적인 딥러닝 기반 객체 검출 모델 중 YOLOv10-L과 RT-DETR-L을 실험에 적용하고 성능을 비교 분석하였다.

실험 결과, RT-DETR-L 모델은 YOLOv10-L 대비 전반적으로 우수한 성능을 보였다. mAP@0.5 기준으로 YOLOv10-L은 0.748, RT-DETR-L은 0.809의 성능을 기록하였고, 클래스별 Precision-Recall 곡선 및 정규화된 혼동행렬 분석에서도 RT-DETR-L 모델이 전반적으로 높은 정확도와 낮은 오탐지율을 유지함을 확인할 수 있었다. 또한, RT-DETR-L의 추론 시간은 이미지 당 평균 10.0ms, YOLOv10-L는 11.4ms으로 측정되어 RT-DETR-L가 더 높은 정확도를 제공하면서도 실시간 처리가 가능한 수준의 응답 속도를 확보함을 알 수 있다. 다만, 학습 시간 측면에서는 YOLOv10-L가 약 6시간 45분, RT-DETR-L는 약 8시간 55분이 소요되어 상대적으로 더 많은 연산 자원이 요구되었다.

Transformer 기반 인코더-디코더 구조를 채택한 RT-DETR-L은 객체와 배경 간 경계 구분과 복잡한 배경 환경에서도 우수한 불량 검출 성능을 보였다.

모델 성능 향상을 위해 RT-DETR-L을 대상으로 데이터 증강 관련 하이퍼파라미터 튜닝 실험도 병행하였다. `hsv_h`, `hsv_s`, `hsv_v`, `flipud`, `mixup`, `copy_paste`, `auto_augment` 등 7가지 주요 증강 항목을 조합하여 수십 회의 실험을 수행하였으며, 그중 성능이 우수한 상위 3개의 설정을 선정하였

다. 특히 RandAugment를 적용한 실험 설정에서는 mAP@0.5가 0.973으로 측정되어 Default 설정 대비 약 16.4%p 향상된 성능을 보였고, 기존에 정밀도가 낮았던 t2, t4 클래스에서도 성능이 크게 개선되었다. 실험 2에서는 t1~t5 모든 클래스에서 0.9 이상의 정확도를 달성하여 클래스 간 불균형 문제 또한 완화되었음을 확인할 수 있었다.

한편, 실험에 사용된 PCB 데이터셋 구성상 일부 한계점도 존재한다. 실제 불량 시료를 수급받는 데 어려움이 많아 불량 이미지 수가 부족하여 전체 6,724장의 데이터 중 약 3,060장은 솔더크림이 미도포된 불량 이미지가 아닌, 임의의 Bare PCB 이미지를 불량으로 간주하여 사용하였다. 또한 RT-DETR과 같은 대규모 모델을 효과적으로 학습시키기에는 약 6천여 장의 데이터가 충분하지 않을 수 있으며, 이에 따라 일반화 성능의 한계 가능성도 내포되어 있다.

아울러, 솔더 페이스트 미도포 불량 중 일부는 실제 패드 경계가 흐릿하거나 손상이 미세하여 시각적으로도 불분명한 경우가 존재하였으며, 이로 인해 모델이 해당 영역을 검출하지 못하고 누락(FN)하는 사례도 있었다. 이는 학습 데이터의 수량뿐만 아니라, 시각적 품질과 경계 명확성도 성능에 영향을 미치는 요소임을 시사한다.

이러한 점은 향후 실제 다양한 미세 결함을 가진 불량 데이터를 충분히 확보하거나, 데이터 생성 기술을 병행하는 방식과 함께, 불량 영역의 경계 명확성을 보완하는 주석 기준 개선 등의 방식을 통해 보완이 필요하다.

이 연구의 목적은 단순한 모델 비교에 그치지 않고, 실제 소규모 SMT 제조 현장에 실질적인 비전을 제공하는 데 있다. 일반적으로 고가의 SPI 장비를 모든 라인에 설치하는 것은 소재·부품·장비 중소기업 입장에서 경제적으로 부담

이 크다. 이에 본 연구는 RT-DETR 모델과 최적화된 데이터 증강 방식을 응용하여, SPI 장비 설치가 어려운 생산 라인에서도 SPI 기능을 지원할 수 있게 끔 고해상도 카메라 기반 납땜 불량 검출 응용 모듈의 구현을 목표로 했다. 추후에는 해당 모델을 임베디드 시스템 또는 엣지 디바이스에 탑재하여 경량화 및 실시간 반응 속도를 확보하고, 다양한 PCB 유형에 대한 확장성과 일반화 성능을 확보하는 것을 과제로 삼을 계획이다.

참고 문헌

[1] Prasad, R.(2013), *Surface Mount Technology: Principles and Practice*, Cham: Springer.

[2] Chen, I.-C., R.-C. Hwang, and H.-C. Huang(2023), "PCB Defect Detection Based on Deep Learning Algorithm," *Processes*, **11**(3), 775. https://doi.org/10.3390/pr11030775.

[3] Ulger, F., S. E. Yuksel, A. Yilmaz, and D. Gokcen(2023), "Solder Joint Inspection on Printed Circuit Boards: A Survey and a Dataset," *IEEE Transactions on Instrumentation and Measurement*, **72**, 2515121.

[4] Siemens Software(2020), *PCB 101 Academy—Learn How Printed Circuit Boards Are Assembled*, Accessed: Feb. 5, 2022, [Online]. Available: https://www.youtube.com/watch?v=usmDfGM2KmU.

[5] Karanjkar, N., A. Joglekar, S. Mohanty, V. Prabhu, D. Raghunath, and R. Sundaresan(2018), "Digital Twin for Energy Optimization in an SMT-PCB Assembly Line," in *Proceedings of the 2018 International Conference on Internet of Things and Intelligent Systems*, Indian Institute of Science, Bangalore, India.

[6] Ho, S. L., M. Xie, L. C. Tang, K. Xu, and T. N. Goh(2001), "Neural Network Modeling with Confidence Bounds: A Case Study on the Solder Paste Deposition Process," *IEEE Transactions on Electronics Packaging Manufacturing*, **24**(4), 323 - 332.

[7] Yen, H. N., D. M. Tsai, and J. Y. Yang(2006), “Full-field 3-D Measurement of Solder Pastes Using LCD-based Phase-shifting Techniques,” *IEEE Transactions on Electronics Packaging Manufacturing*, **29**(1), 50 - 57.

[8] Hui, T. W. and G. K. H. Pang(2009), “Solder Paste Inspection Using Region-based Defect Detection,” *International Journal of Advanced Manufacturing Technology*, **42**(7 - 8), 725 - 734.

[9] Ye, R., M. Chang, C.-S. Pan, C. A. Chiang, and J. L. Gabayno(2018), “High-Resolution Optical Inspection System for Fast Detection and Classification of Surface Defects,” *International Journal of Optomechatronics*, **12**(1), 1 - 10.

[10] Deng, Y.-S., A.-C. Luo, and M.-J. Dai(2018), “Building an Automatic Defect Verification System Using Deep Neural Network for PCB Defect Classification,” in *Proceedings of the 4th International Conference on Frontiers of Signal Processing (ICFSP)*, Sep. 2018, 145 - 149.

[11] Li, J., J. Gu, Z. Huang, and J. Wen(2019), “Application Research of Improved YOLO v3 Algorithm in PCB Electronic Component Detection,” *Applied Sciences*, **9**(18), 3750.

[12] Hu, B. and J. Wang(2020), “Detection of PCB Surface Defects with Improved Faster R-CNN and Feature Pyramid Network,” *IEEE Access*, **8**, 108335 - 108345.

[13] Chauhan, A. P. S. and S. C. Bhardwaj(2011), “Detection of Bare PCB Defects by Image Subtraction Method Using Machine Vision,” in *Proceedin*

gs of the World Congress on Engineering 2011 (WCE 2011), Vol. II, July 6 - 8, London, U.K., ISBN: 978-988-19251-4-5.

[14] Ma, J.(2017), "Defect Detection and Recognition of Bare PCB Based on Computer Vision," in *Proceedings of the 36th Chinese Control Conference*, July 26 - 28, Dalian, China.

[15] Zhu, W., H. Gu, and W. Su(2020), "A Fast PCB Hole Detection Metho d Based on Geometric Features," *Measurement Science and Technology*, **31**, 095402. https://doi.org/10.1088/1361-6501/ab8b21.

[16] Annaby, M. H. and Y. M. Fouda(2019), "Improved Normalized Cross-C orrelation for Defect Detection in Printed-Circuit Boards," *IEEE Transactio ns on Semiconductor Manufacturing*, **32**(2), 199.

[17] Krizhevsky, A., I. Sutskever, and G. E. Hinton(2012), "ImageNet Classi fication with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems (NeurIPS)*, Lake Tahoe, NV, USA, 1097 - 1105.

[18] Loo, M. C., Z. A. A. Salam, and R. Logeswaran(2023), "CNN Aided S urface Inspection for SMT Manufacturing," in *Proceedings of the 15th Inte rnational Conference on Developments in eSystems Engineering (DeSE)*, Kuala Lumpur, Malaysia.

[19] Sezer, A. and A. Altan(2021), "Detection of Solder Paste Defects with an Optimization-Based Deep Learning Model Using Image Processing Tech niques," *Soldering & Surface Mount Technology*, **33**(4), Article published:

August 12; Issue published: October 19.

[20] Chen, W., Z. Huang, Q. Mu, and Y. Sun(2022), “PCB Defect Detection Method Based on Transformer-YOLO,” *IEEE Access*, **10**, 132506 - 132515. https://doi.org/10.1109/ACCESS.2022.3228206.

[21] Felzenszwalb, P. F., R. B. Girshick, D. McAllester, and D. Ramanan(2010), “Object Detection with Discriminatively Trained Part-Based Models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **32**(9), 1627 - 1645.

[22] Girshick, R., J. Donahue, T. Darrell, and J. Malik(2014), “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 580 - 587.

[23] Redmon, J., S. Divvala, R. Girshick, and A. Farhadi(2016), “You Only Look Once: Unified, Real-Time Object Detection,” *arXiv preprint arXiv:1506.02640v5*, https://arxiv.org/abs/1506.02640.

[24] Wang, A., H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding(2024), “YOLOv10: Real-Time End-to-End Object Detection,” arXiv preprint arXiv:2405.14458v2, https://arxiv.org/abs/2405.14458.

[25] Akhmedov, F., R. Nasimov, and A. Abdusalomov(2023), “Dehazing Algorithm Integration with YOLO-v10 for Ship Fire Detection,” *Sensors*, **23**(19), 8216. https://doi.org/10.3390/s23198216.

[26] Zhao, Y., W. Lv, S. Xu, J. Wei, G. Wang, Q. Dang, Y. Liu, and J. Ch

en(2023), “DETRs Beat YOLOs on Real-Time Object Detection,” *arXiv pre print arXiv:2304.08069*, https://arxiv.org/abs/2304.08069.

[27] Jocher, G.(2023), “YOLOv8,” *Ultralytics GitHub Repository*, https://git hub.com/ultralytics/ultralytics/tree/main.

[28] Carion, N., F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zag oruyko(2020), “End-to-End Object Detection with Transformers,” in *Procee dings of the European Conference on Computer Vision (ECCV)*, 213 - 229. Springer.

[29] Zhu, X., W. Su, L. Lu, B. Li, X. Wang, and J. Dai(2020), “Deformable DETR: Deformable Transformers for End-to-End Object Detection,” in *Pro ceedings of the International Conference on Learning Representations (ICL R)*.

[30]Lv, W., Y. Zhao, Q. Chang, K. Huang, G. Wang, and Y. Liu(2024), “RT -DETRv2: Improved Baseline with Bag-of-Freebies for Real-Time Detectio n Transformer,” *arXiv preprint arXiv:2407.17140*, https://arxiv.org/abs/2407. 17140.

[31] Zhang, Y., J. Song, X. Yu, and X. Ji(2025), “WMC-RTDETR: A Light weight Tea Disease Detection Model,” *Frontiers in Plant Science*, **16**, 1574 920. https://doi.org/10.3389/fpls.2025.1574920.

[32] Cubuk, E. D., B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le(2019), “AutoAugment: Learning Augmentation Policies from Data,” *arXiv preprint arXiv:1805.09501*, https://arxiv.org/abs/1805.09501.

[33] Cubuk, E. D., B. Zoph, J. Shlens, and Q. V. Le(2019), “RandAugment: Practical Automated Data Augmentation with a Reduced Search Space,” *arXiv preprint arXiv:1909.13719*, https://arxiv.org/abs/1909.13719.

[34] Hendrycks, D., N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan(2020), “AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty,” *arXiv preprint arXiv:1912.02781*, https://arxiv.org/abs/1912.02781.

[35] Raschka, S.(2023), “Comparing Different Automatic Image Augmentation Methods in PyTorch,” *Ahead of AI Magazine*, January 29. https://sebastianraschka.com/blog/2023/autoaugment-comparison.html