2023 전기 졸업과제 중간보고서

인공지능 비전 기반 RT필름 결함 자동판독 기술

분과 C

팀 번호 28

지도교수 김 종 덕 교수

팀 장 201824170 정다현

조 원 201714532 이현규

조 원 201824640 지민철

목차

1. 연구 목적과 변경 사항

1-1. 초기 연구 목적 및 변경된 연구 목적

초기 연구 목적은 RT 필름을 활용하여 산업용 비 파괴 검사에서 소재의 결함을 감지하고 정확한 위치를 파악하여 소재의 내구성과 안전성을 평가하는 것이었다. 이를 위해 인공지능 비전 기반의 RT 필름 결함 자동 판독 모델을 제작하여 필름을 빠르게 검사하여 좀 더 효율적인 결함 탐지를 하는 것이 목적이었다

하지만, 이미지의 총량이 적고 촬영된 이미지가 불규칙적이었다. 결함 이미지는 77장, 정상 이미지는 148장으로 추가로 생성해 낸 이미지 (이하 fake 이미지)를 포함해도 결함 이미지는 214장, 정상 이미지는 249장으로 충분치 못한 데이터였다. 또한 관의 종류가 3가지였다. 각각을 나누어서 생각해 보면 더욱 데이터는 줄어든다.

대형관 결함 이미지는 71(+38)장, 정상 이미지는 38(+68)장이다. 중형관 결함 이미지는 4(+94)장, 정상 이미지는 79(+22)장이다. 소형관은 결함 이미지가 2(+5)장, 정상 이미지 31(+11)장이다. 괄호 안 숫자는 fake 이미지의 숫자이다. Fake 이미지는 임의적으로 만들어낸 이미지이기에 데이터가 오염되었을 확률이 높고, 이로 인해 모델학습에서 overfitting 등의 문제가 발생할 가능성이 높다. Overfitting의 문제가 발생하면 실제 이미지를 입력데이터로 넣었을 때, 결함을 제대로 검출하지 못할 수 있기 때문에 fake 이미지를 따로 분류하였다.

따라서 총 이미지의 양도 적지만 중형관과 소형관의 이미지는 더욱 적은 것을 볼 수 있다. 또한 결함의 종류도 여러 가지고, 그중 일부 결함은 전체에서 1개, 2개 정도만 있어 그러한 결함을 학습시켜 탐지해 내는 것은 매우 힘들어 보인다. 또한 RT 필름 이미지 자체도 규격화 되어 있지 않고 결함을 탐지해야 하는 용접선 위치도 제각각이여서 더욱 결함을 찾기 어려워 보인다.

이에 논의 끝에, 결함 탐지는 후순위로 미루기로 하고, 결함을 발견해야 하는 용접선 위치를 찾아낼 수 있는 모델을 우선적으로 제작하기로 하였다. 따라서 변경된 연구 목적은 다음과 같다.

용접관의 용접선 위치를 탐지하는 모델 개발을 목표로 한다. 이 모델을 통해 사람이 직접 검사할 때도 결함을 찾기 쉽게 하고, 후에 이 모델을 바탕으로 결함 탐지를 위한 모델을 개발하기 위한 초석으로 삼으려 한다.

.

1-2. 그 외 변경된 점

착수보고서에서는 YOLO(You Only Look Once)를 사용하여 결함을 탐지하고자 했다. 하지만 과제를 위한 연구를 지속한 결과, YOLO는 이 과제를 해결하기 위한 적합한 모델이 아니었다.

YOLO는 이미지를 N x N Grid cell로 나누고 각 cell마다 하나의 객체를 예측하여, 미리 설정된 개수의 BBox (Boundary Box)를 통해 객체의 위치와 크기를 파악하는 모델이다.

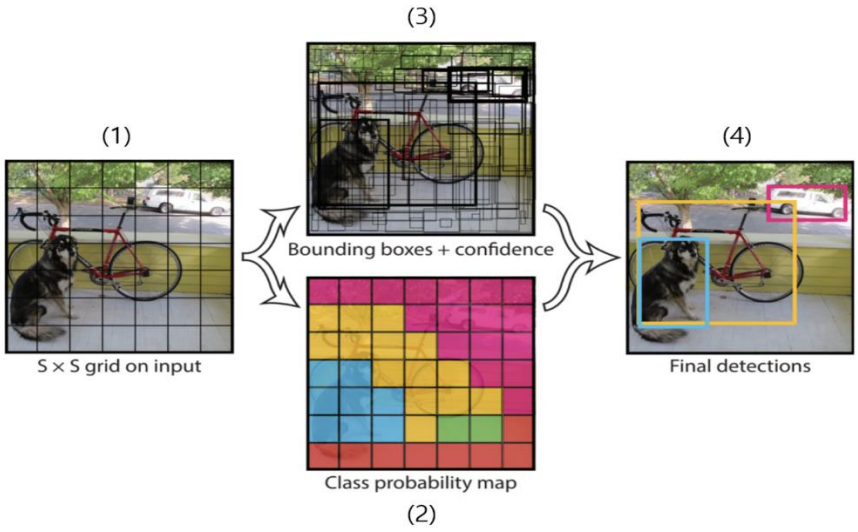


그림 1 YOLO의 작동 방식

위 그림 1처럼 YOLO는 각 그리드 영역에 대해서 BBox와, BBox에 대한 신뢰도 점수를 예측한다. 그에 더해 분류 작업이 동시에 일어난다. 그 후에 NMS 알고리즘[1]을 통해 이미지를 선별하여 그림 1 내의 (4)처럼 객체를 탐지하게 된다.

하지만 이런 방식은 객체를 파악할 때, 객체가 겹쳐 있으면 몇몇 객체를 탐지 못 할 가능성이 있다. 또한, 작은 물체에 대한 탐지 성능도 낮다. 객체가 크다면 BBox 간 IoU[2] 값 차이가 커져서 적절하게 물체 탐지를 할 수 있지만, 객체가 작

다면 Bbox 간 IoU 값 차이가 작아 적절한 결과가 나오기 어렵다.

그래서 YOLO는 큰 객체를 classification하고 식별하기 위해 사용되는데, 이는 현재 우리가 작업해야 할 불량을 탐지하는 과정과는 맞지 않다. 따라서 착수 보고서에서 YOLO를 사용하겠다고 했지만, 이를 변경하고자 한다

1) **NMS 알고리즘: 특정 객체를 과도하게 식별하는 것을 방지하기 위해 가장 높은 확률인 BBox 를 선택하는 알고리즘이다. 이 알고리즘은 YOLO 가 아래 그림 2 처럼 만들어낸 BBox 중에 가장 확률이 높은 BBox 를 선택하여 오른쪽과 같은 결과를 보여주게 된다.**

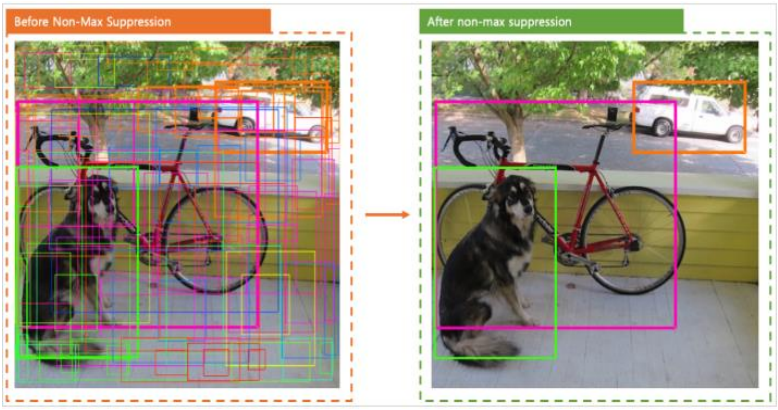


그림 2 NMS 알고리즘

2) **IoU: BBox 의 위치 정확도에 대한 평가 도구이다, 즉 특정한 Bbox 의 IoU 값이 근처의 BBox 보다 높다면 그 BBox 가 가장 적절한 것이다.**

2. 설계 상세화

2-1 데이터 전처리

다음은 모델에 이미지를 학습하기 전에 효율적인 학습을 위해 이미지를 전처리 했던 과정이다.

그림 3 원본 이미지

그림 3은 아무런 가공을 거치지 않은 원본 이미지로 **그림 3**을 사용해 어떤 전처리를 했는지 예시를 보여주고자 한다.

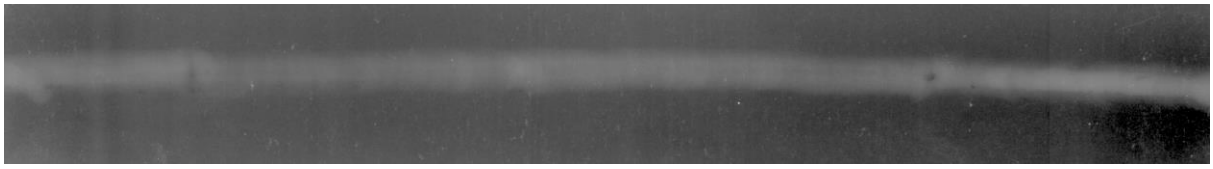


그림 4 Resize한 이미지

용접선이 위치가 일관적이지 않아 최대한 용접선을 포함하면서 배경을 제거할 수 있는 사이즈로 [550:1100, 50:4300]를 선택하여 이미지를 resize하여 **그림 4**를 얻었다.

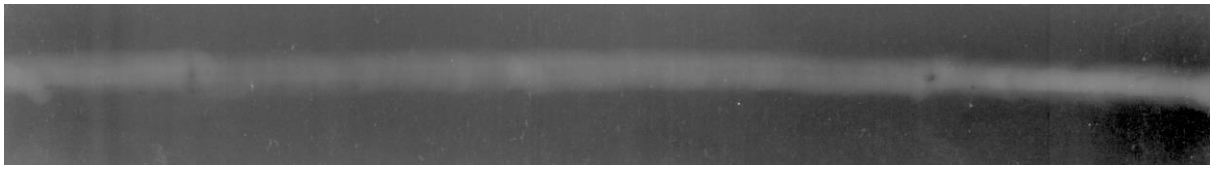


그림 5 Grayscale 변환한 이미지

이후 이미지를 grayscale 변환하여 **그림 5**를 얻었다. 이는 이미지를 0과 255까지의 단일 채널 값으로 표현하는 것을 의미한다. 제공된 필름 이미지는 이미 grayscale처럼 보이지만 실제로는 3채널 값을 포함하고 있다. 후에 threshold라는 전처리 방법을 사용하기 위해서는 변환이 필수적이고, 단일 채널로 변환됨으로써 후에 딥러닝 모델에서 처리할 때 속도 향상도 꾀할 수 있다.

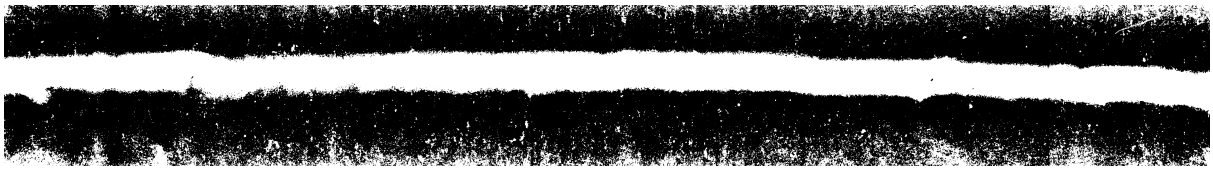


그림 6 Threshold 변환한 이미지

이후 Threshold 변환을 통해 **그림 6**을 얻었다. Threshold 변환이란 임곗값을 기준으로 이하의 픽셀은 0으로, 이상의 픽셀은 255로 이미지를 이진화하는 것이다. 이를 통해 두 가지 값으로 구분하여 간단한 객체 분할이 가능하다. 임의의 임계값 하나만 사용하면 이미지의 조명이 일정하지 않거나 배경색이 나뉘는 경우에 좋은 결과를 얻지 못하기 때문에, 임곗값은 이미지를 여러 영역으로 나눈 후에 그 주변 픽셀값을 기준으로 계산하는 adaptiveThreshold를 사용하여 정하였다.

그림 7 Eroding 연산 설명 1

최종 결과 이미지

그림 8 Eroding 연산 설명 2

이후 Eroding 연산을 수행했다. **그림 7과 8**을 보면 Eroding 연산이 어떤 식으로 작동하는지 알 수 있다. 구조 요소(element)를 활용해 이웃한 픽셀을 최소 픽셀 값으로 대체한다. 이 eroding 연산을 적용하면 밝은 영역이 줄어들고 어두운 영역이 늘어난다. 이미지의 노이즈를 줄이고자 사용하였다. 그림 9를 얻을 수 있었다.

그림 9 Eroding 연산 후 이미지

이후 Dilation 연산을 수행했다. 구조 요소(element)를 활용해 이웃한 픽셀들을 최대 픽셀값으로 대체한다. Dilation 연산을 적용하면 어두운 영역이 줄어들고 밝은 영역이 늘어난다. eroding의 반대 연산이며, 노이즈 제거 후 줄어든 크기를 복구하고자 사용하였다. **그림 10**을 얻을 수 있었다.

그림 10 Dilation 연산 후 이미지

contours 함수를 이용해 용접선과 배경에 경계선을 그었다. Contours 함수는 일

정한 값을 가지는 점들을 연결하여 곡선을 나타내는 함수이다. 이미지의 중앙에서부터 시작하여 지도의 등고선처럼 영역과 우선순위를 구분하여 처리하게 된다. 전처리할 원본 이미지들의 용접선이 중앙에서 시작되고, 용접선과 배경 부분을 구분하고자 사용하였다.

아래 **그림 11**을 보면 contours 함수가 어떻게 작동하는지 알 수 있다. 그림 11의 contours는 중앙의 0에서 시작하여 8까지, 총 9개의 등치선을 가진다.

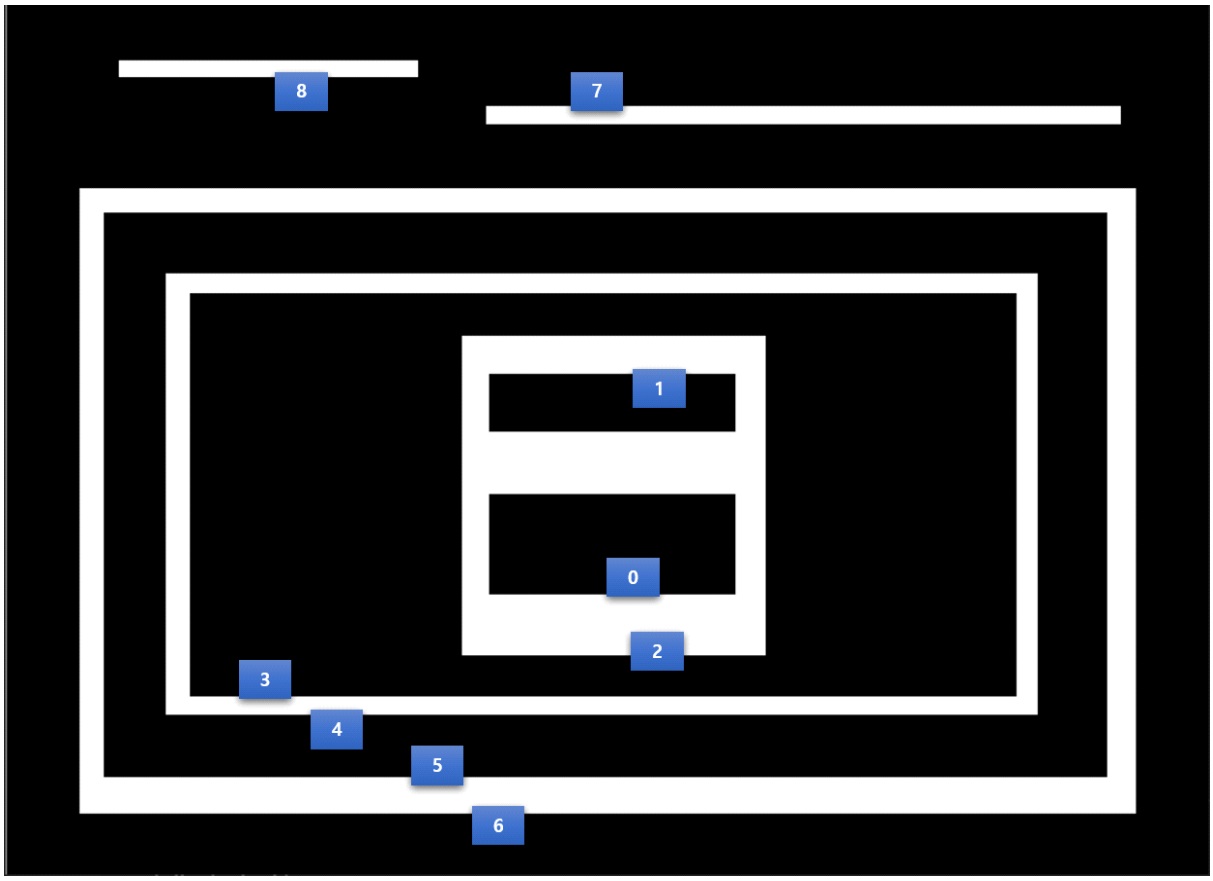


그림 11 contours 예시 이미지

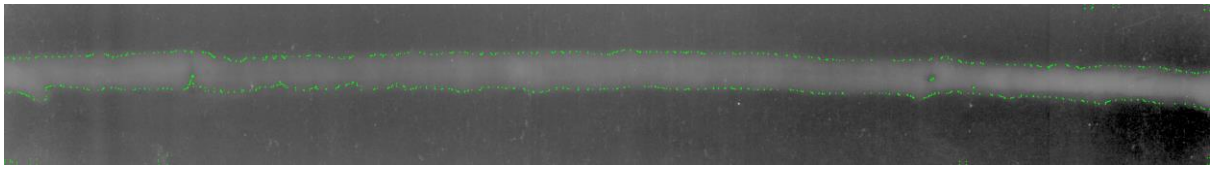


그림 12 contours 변환 후, 최종 변환 이미지

결과로 **그림 12**를 얻을 수 있었다.

이와 같은 전처리를 수행하였지만, 다양한 문제점들이 존재한다.

그림 13 원본 이미지와 예외 이미지

우리는 고정된 사이즈 [550:1100, 50:4300]으로 리사이즈 하였다. 대부분의 이미지가 해당 영역 안에 포함되지만 몇몇 이미지는 포함되지 않는다. 또한 이후에 추가적으로 받게 될 이미지도 용접관의 위치가 고정된 영역 안에 포함될지 확신할 수 없다.

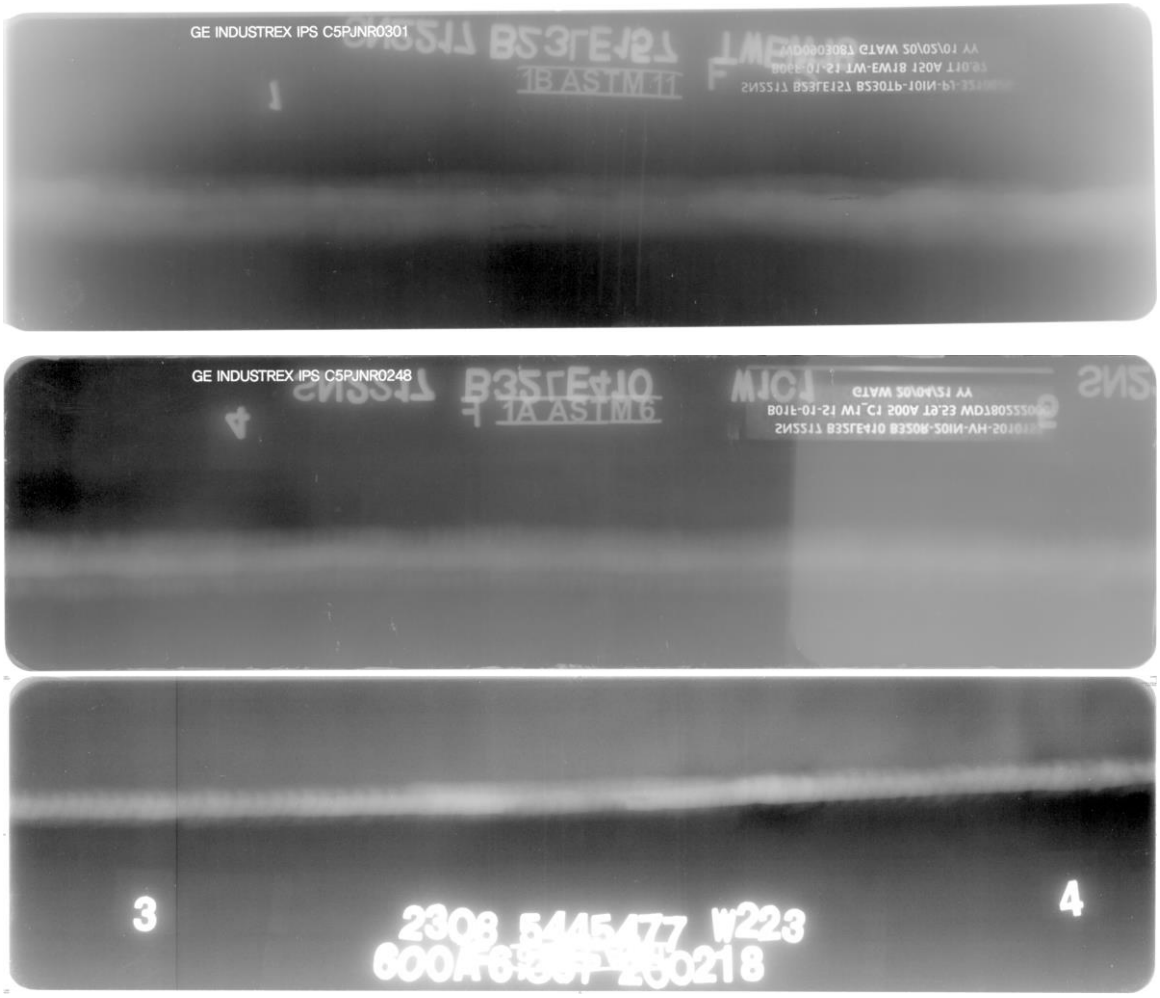


그림 14 이미지 배경의 차이

그림 15 결함 이미지와 정상 이미지

제공받은 이미지는 grayscale으로 결함은 검은색, 배경은 회색, 용접관은 흰색으로 나타나지만, 몇몇 이미지들은 위와 같이 용접관을 흰색으로 덮거나, 검은색 영역이 용접관에 인접하는 등 전처리하기 힘든 배경을 가지고 있다. 위와 같은 이미지에서 threshold를 사용할 경우 흰색 배경을 용접관으로, 검은색 배경을 결함으로 인식할 수 있다.

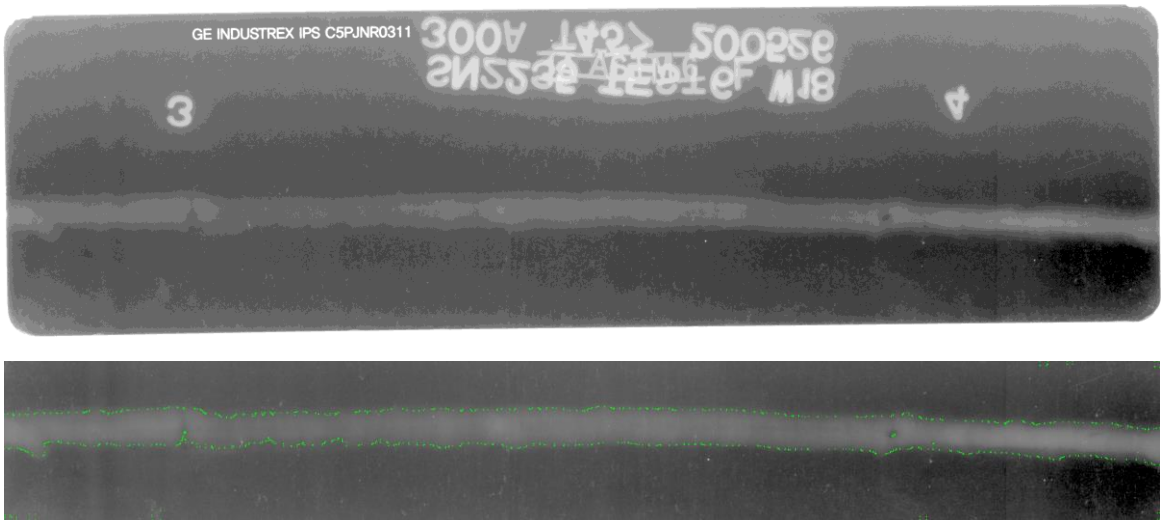


그림 16 원본 이미지와 최종 변환 이미지

이와 같은 문제점 외에도 전처리된 이미지를 학습시키는 것이 결함 검출 학습에 도움이 되지 않을 것이다. 근본적으로 이미지가 다르기 때문이다. 매번 입력되는 이미지를 전처리하면 되지만 많은 자원이 소모되고 공장에서 제공되는 이미지는 동일한 용접선의 위치를 장담할 수 없다. 그렇기 때문에 이와 같은 전처리를 하는 것이 아닌 원본 이미지를 라벨링하여 학습시키는 방법으로 모델을 제작하기로 했다.

하지만, 이와 같은 전처리를 수행하였지만, 문제점들이 몇 가지 존재했다. 용접선 위치가 바뀔 수 있고, 전처리된 이미지를 학습시켜도 결함 검출 학습에 도움이 되지 않을 수 있었다. 매번 입력하는 이미지를 전처리하면 되지만 많은 자원이 소모된다. 공장에서 제공하는 필름 이미지에서는 동일한 용접선의 위치를 장담할 수 없다. 그렇기 때문에, 원본 이미지를 라벨링하여 학습시키는 방법으로 모델을 제작하기로 했다.

2-2 딥러닝 모델 개요

YOLO를 사용하지 않기로 했으므로 다른 적절한 모델을 선정하는 것이 필요하다. 조사를 통해 모델 후보를 골라내었고 그 중에 하나를 선정하기로 했다.

다음은 모델 후보들의 간략한 개요이다.

RNN

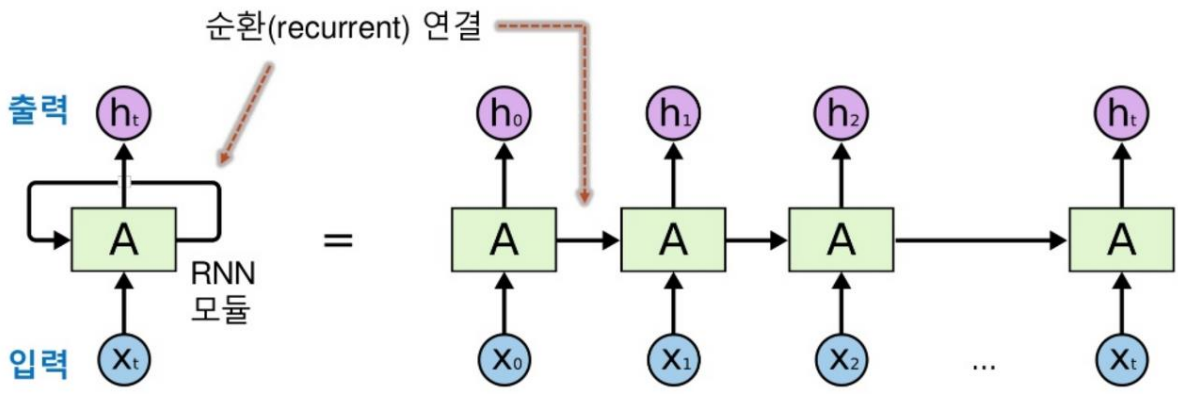


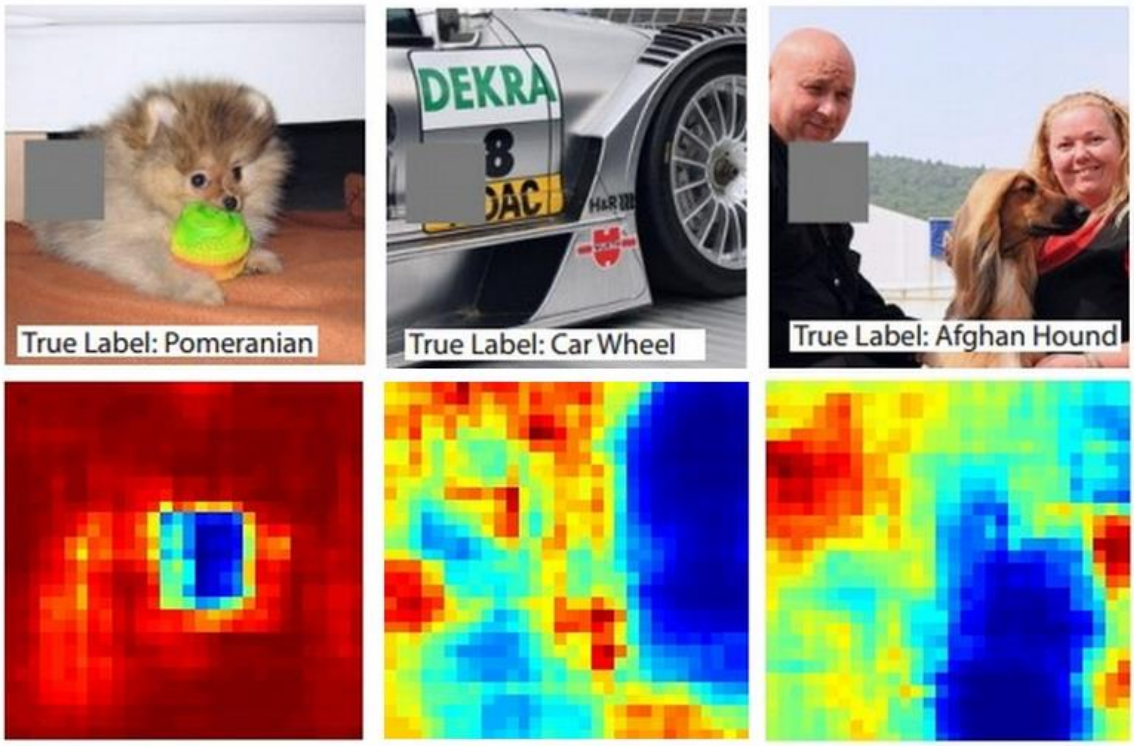
그림 17 순환 연결

RNN은 Recurrent Neural Network의 약자로, 순환 신경망이라고도 불린다. 시퀀스 데이터(Sequence data)를 처리하는데 사용된다. RNN은 입력 데이터와 이전 단계에서의 출력을 함께 고려하여 현재의 출력을 계산하는 방식을 갖추고 있다. 즉, **그림 17**에서 h_n을 계산할 때는 h_{n-1}을 참조하여 계산한다. 일반적인 인공신경망은 입력과 출력 사이에는 연결이 단방향으로만 이루어지지만, RNN은 순환적인 구조를 가지며 이전 단계의 출력이 다음 단계로 입력되는 방식으로 동작한다. 이로 인해 RNN은 시퀀스 데이터를 처리하면서 데이터 간의 시간적인 의존성을 파악할 수 있어, 자연어 처리와 음성 인식 등의 시퀀스 데이터 처리에 많이 활용된다.

CNN

CNN은 Convolutional Neural Network의 약자로, 합성곱 신경망이라고도 불린다. 주로 이미지 처리에 사용되는 딥러닝의 한 종류이다. CNN은 입력 이미지에 대해 지역적인 특징을 감지하고 이러한 특징들을 조합하여 이미지를 분류하거나 인식하는 데에 사용된다.

CNN은 보통 Convolutional Layer와 Pooling Layer으로 구성된다. Convolutional Layer에서는 이미지에 여러 개의 작은 필터를 적용하여 아래 **그림 18**처럼 Feature Map을 생성한다. 이러한 필터는 이미지의 지역적인 특징, 예를 들면 경계, 질감, 색상 등을 감지하는 역할을 수행한다. Pooling Layer에서는 Feature Map을 축소하여 계산을 줄이고, 불필요한 정보를 제거하여 추상화된 특징을 유지한다.



Reference: Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818-833)

그림 18 CNN의 Feature Map

CNN은 이러한 Convolutional Layer와 Pooling Layer을 여러 층으로 쌓아 올려서 이미지의 High-level feature들을 추출하는 과정을 반복한다. 이 High-level feature 들은 Fully Connected Layer으로 전달되어 최종적으로 이미지의 분류 또는 인식을 수행한다.

Faster R-CNN

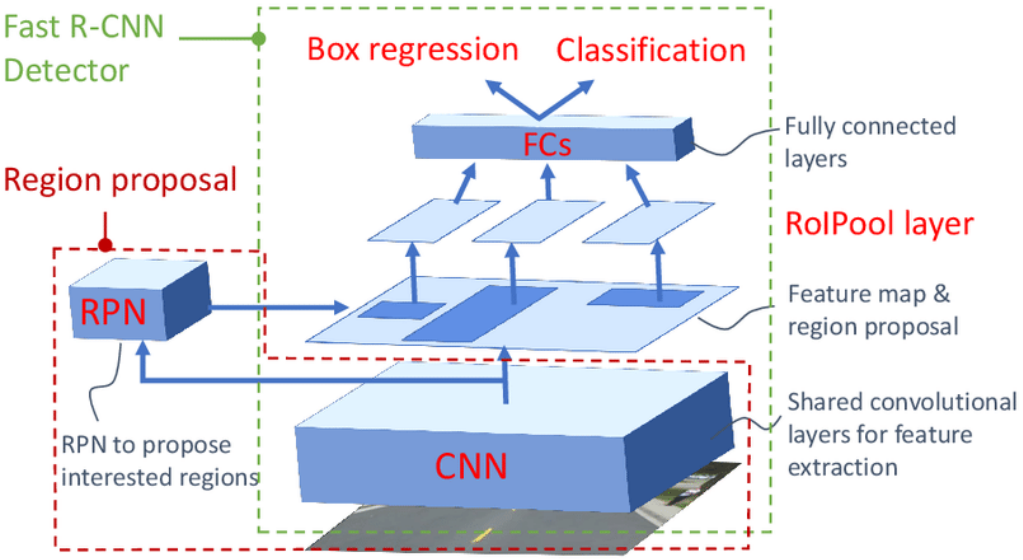


그림 19 Faster R-CNN의 구조

Faster R-CNN은 **그림 19**처럼 R-CNN(Regions with CNN features)의 개선된 형태인 Fast R-CNN과 Region Proposal Network(RPN)가 합쳐진 형태이다. R-CNN은 물체 후보 영역(Region Proposal)을 먼저 생성하고, 이후에 각 영역에 대해 딥러닝 모델을 적용하여 물체를 검출하는 형태의 모델이다. Fast R-CNN은 여기에 Region of Interest (RoI) Pooling 기법을 도입하여, 물체 후보 영역을 따로 추출하는 것이 아니라 딥러닝 모델에 입력 이미지를 전체로 넣고 물체 후보 영역에 대한 계산을 병렬로 수행하여 R-CNN보다 빠른 속도를 가질 수 있다.

Faster R-CNN은 여기에 RPN을 도입하여 Region Proposal을 개선한 버전이다. RPN은 이미지 내의 다양한 위치와 크기에 대해 물체 후보 영역을 자동으로 예측한다. 이후 RPN에서 생성한 영역들에 대해 딥러닝 모델을 적용하여 물체를 검출한다. 이렇게 하나의 통합된 모델로 물체 검출과 Region Proposal을 동시에 수행하여 Faster R-CNN은 Fast R-CNN보다 더 빠른 속도를 가진다.

그 외

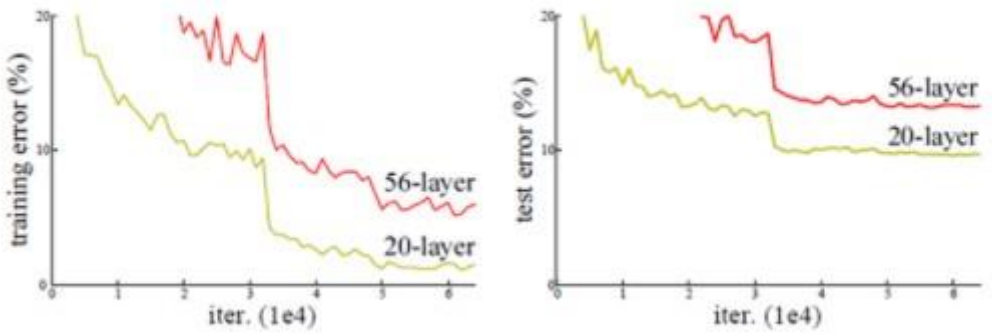
이외에 **UNet** 과 **ResNet** 이 있는데, 이는 최종 후보에 선정된 모델로서 2-3 에서 그 특징들을 정리했다.

2-3. 최종 선정 후보 및 선정

적은 데이터로도 충분한 결과가 나올 것, 작은 크기도 탐지할 수 있을 것, 가용할 수 있는 컴퓨터 자원으로 학습할 수 있을 것, Image Segmentation을 수행할 수 있을 것 등 몇 가지 상황을 고려한 결과, 적합한 모델의 후보를 2개까지 줄일 수 있었다. U-net과 ResNet이다. 이를 자세히 살펴보고 두 모델 중 하나를 선택하기로 했다.

2-3-1. ResNet

(1) 배경 및 원리



그래프 1 Layer 수에 따른 오류

CNN이 이미지 인식 분야에서 뛰어난 성능을 보여주지만, 네트워크 layer를 깊게 쌓게 된다. 일반적인 모델에서 Layer가 많아지면 성능이 향상된다. 하지만 항상 그런 것만 은 아니다. 위의 표는 Resnet을 발표한 Microsoft 팀의 논문 Deep Residual Learning for Image Recognition에 나오는 표인데 층이 많아질수록 오류가 많아지는 것을 볼 수 있다. 일반적인 네트워크는 입력 데이터가 여러 개의 층을 통과하면서 변환되어 출력을 생성한다. 그러나 이러한 변환 과정에서 정보의 손실이 발생할 수 있다. 특히 층이 매우 많다면 gradient[1)]가 소실되거나 폭주하여 효과적인 학습이 어려워질 수 있다. 그렇기

[1] **Gradient: 함수의 기울기를 나타내는 개념으로, Deep Neural Network에서 가중치 및 편향과 같은 매개변수를 업데이트하는 데 사용된다. Deep Neural Network는 입력 데이터를 통과하여 출력을 생성하는데,**

에 매우 많은 층은 오히려 더 나쁜 결과를 갖게 된다.

ResNet은 이러한 문제를 해결하기 위해 잔차 연결(residual connection)을 도입한다. 잔차 연결은 입력과 출력 사이에 스킵 연결(skip connection)을 생성하여 잔차를 직접 학습하는 구조를 구성한다. 입력 데이터를 더하여 원래의 입력을 유지하면서 중간층에서의 변환을 학습할 수 있도록 한다. 이렇게 하면 네트워크가 잔차를 학습하고 이를 통해 입력 데이터에 더해 최종 출력을 생성하게 된다.

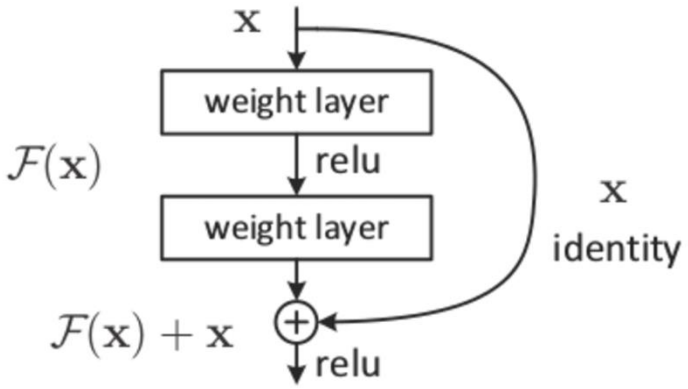


그림 20 스킵 연결

즉 레이어를 통과한 데이터 f(x)+x와 레이어를 통과하지 않은 입력 데이터 x를 비교해 잔차 f(x)를 계산해 낸다.

잔차 연결은 gradient 흐름을 개선하여 깊은 신경망에서의 학습을 원활하게 하며, gradient 소실과 폭주 문제를 완화시킨다. 또한, 적은 데이터에서도 효과적인 학습을 가능하게 하고 과적합을 줄일 수 있는 파라미터의 효율적인 사용을 가능하게 한다.

이 과정에서 네트워크의 가중치와 편향이 사용된다. 훈련 단계에서, 네트워크의 출력과 정답(label) 간의 차이를 측정하는 손실 함수(loss function)가 정의된다. 이 손실 함수를 최소화하기 위해 역전파(backpropagation) 알고리즘이 사용되는데, gradient를 계산하여 네트워크의 가중치와 편향을 조정한다. gradient는 손실 함수의 값이 가장 작아지는 방향을 가리키며, 역전파 알고리즘을 통해 네트워크의 각 층에서 계산된다. 역전파는 출력부터 입력 방향으로 진행되며, gradient를 계산하기 위해 연쇄 법칙(chain rule)을 사용한다. 각 층에서는 gradient가 이전 층으로 전파되어 가중치와 편향을 업데이트하게 된다.

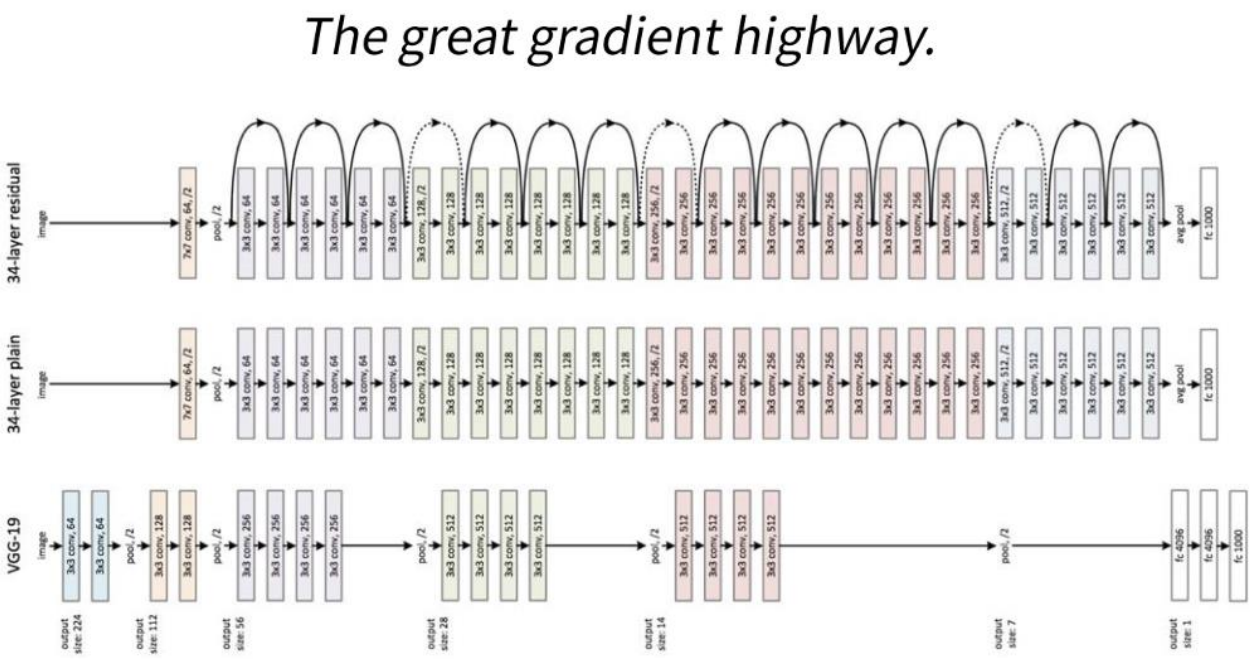
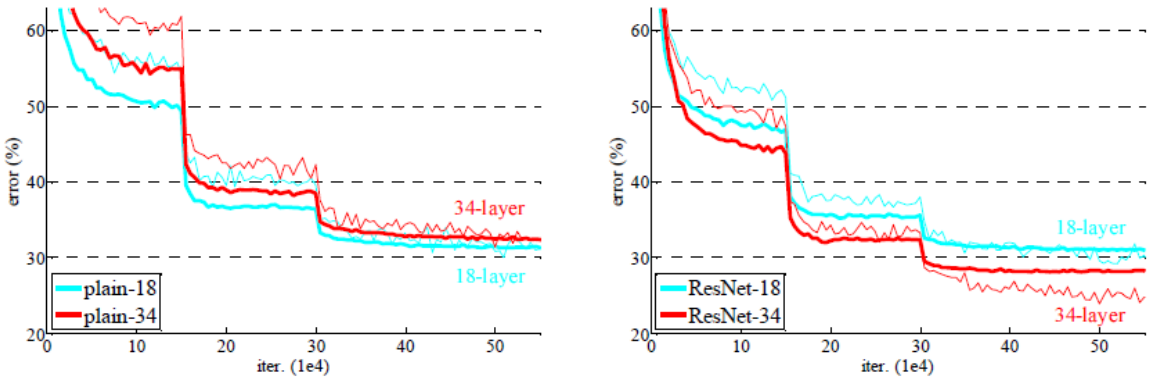


그림 21 ResNet의 구조

그림 14처럼 ResNet은 기본적으로 VGG-19의 구조를 뼈대로 한다. 거기에 Convolution layer들을 추가해서 Layer의 수를 늘린 후에, skip connection을 추가하였다. **그림 14**에서 34-layer Plain(이하 Plain)은 34-layer residual에서 skip connection을 뺀 구조로, skip connection이 없다면 어떤 결과가 나오는지 아래 **그래프 2**에서 볼 수 있다



그래프 2 ResNet과 일반 모델의 층에 따른 성능 비교

왼쪽 Plain과 오른쪽 ResNet을 비교해보면 layer가 많아질 수록 ResNet은 성능이 좋아지지만, Plain은 그렇지 못함을 보여준다.

(2) 평가

ResNet은 현재 우리의 프로젝트에 적용하기 좋은 모델이다. 하지만, 모델 특성상 많은 Parameter를 요구하는데, 이는 현재 노트북으로 개발하는 개발환경에 비추었을 땐, 좋지 못한 모델이다. 그렇기에 조금 더 가벼운 UNet이 낫다고 판단하였다.

3-2-2. UNet

(1) 배경 및 원리

UNet은 CNN 모델의 한계를 극복하기 위해 개발된 딥러닝 아키텍처이다. 기존의 CNN은 Convolution(합성곱) 필터를 사용하여 이미지의 특징을 추출하고, Fully Connected Layer 형태로 특징을 1열로 배치하여 분류 작업에 사용한다. 하지만 이러한 방식은 기본적으로 1차원 행렬에 대해서만 연산이 가능하고, 상세한 특징을 추출하기 어려웠다.

UNet은 이러한 제약을 극복하기 위해 개발되었다. UNet은 Fully Convolutional Network(FCN)의 개념을 적용하여 모든 계층에서 합성곱 연산을 사용합니다. 이로 인해 입력과 출력 이미지의 크기가 다르더라도 효과적인 연산이 가능하며, 상세한 특징을 추출할 수 있게 되었다.

또한, UNet은 인코더-디코더 구조와 skip connection[2)]을 결합하여 더욱 정확한 분할을 가능하게 한다. 인코더에서는 입력 이미지를 점차 축소하고 특징을 추출하며, 디코더에서는 추출된 특징들을 확대하여 출력 이미지를 생성한다. 스킵 연결은 인코더와 디코더 사이에 추가되어, 높은 해상도와 저수준의 정보를 보존한다. 이를 통해 더 정확한 분할 결과를 얻고 빠른 속도의 연산이 가능하게 된다.

[2] **skip connection: Encoder layer와 Decoder layer의 직접 연결**

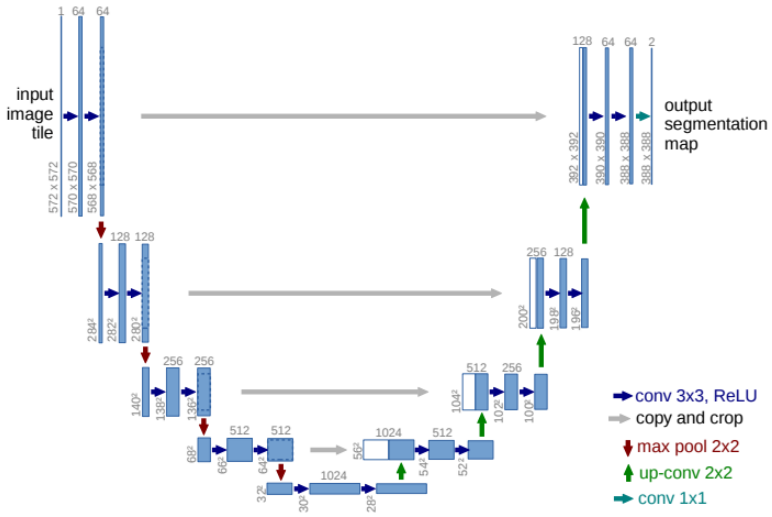


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

그림 22 UNet의 구조

그림 22는 UNet의 구조를 나타낸 것이다. 중앙을 기준으로 크게 2가지 구조로 나뉘는 것을 볼 수 있다.

Contracting Path와 Expanding Path을 사용하는데, Contracting Path에서는 이미지 픽셀을 보며 Context를 추출하고, Feature Map을 위 **그림 22**에서 copy and crop 이라 명시된 Skip connection을 통해 Expanding Path로 보낸다. 이는 Convolution 을 하면서 생기는 border pixel에 대한 정보 손실을 복구하기 위해서이다. Expanding Path에서는 Context와 위치정보를 결합하여 각 Pixel마다 어떤 객체에 속하는지 구분한다.

좀 더 자세한 구조를 알아보자. **[Contracting Path]**에서는 인코딩 단계에서 얻은 feature을 디코딩의 단계의 각 레이어에 합치는 방법을 사용한다.

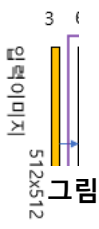


그림 23에서 세로 숫자는 map의 차원을 표시, 가로 숫자는 채널의 수를 의미한다. 따라서 **그림 23**의 입력 이미지의 크기는 512x512이고, 3개의 채널을 가지고 있는 이미지이다.

그림 23

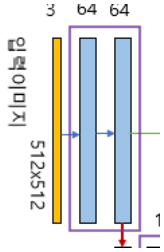


그림 24의 파란색 블록은 한 화살표마다 3x3 Convolution(합성곱), Batch Normalization(정규화), ReLU (활성화 함수)가 실행되는 것을 의미한다.

그림 24

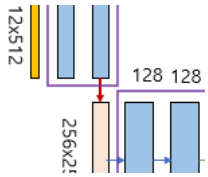


그림 25에서 볼 수 있듯이 다음 스텝으로 넘어 갈 때마다 2x2 max-pooling 연산을 수행한다. 이를 통해 Feature map의 크기가 절반으로 줄어들게 된다. 또한 down-sampling마다 채널의 수가 2배 늘어난다.

그림 25

[Expanding Path]에서는 Context와 위치정보를 결합하여 각 Pixel마다 어떤 객체에 속하는지 구분한다.

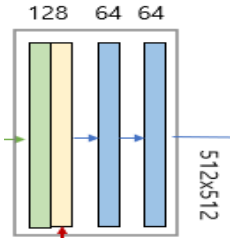
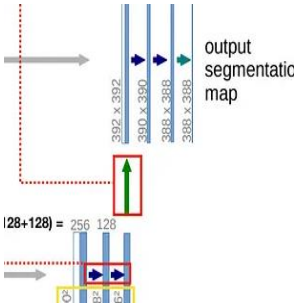


그림 26의 노란 블록은 이전 layer 혹은 bridge[5]로 넘어온 feature map이다. 이는 파란색 화살표(ConvBlock (3x3 convolution, ReLU))를 통해 변환된다.

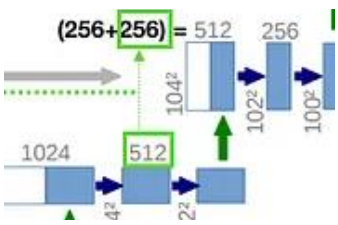
5) bridge: 그림 15의 연결 부위

그림 26



Expanding Path는 **그림 27**에서 볼 수 있듯이, 각 스텝마다 2x2 up-convolution을 수행하여 feature map의 크기가 4배 늘어난다.

그림 27



또한 **그림 28**에서처럼 up sampling마다 채널의 수가 절반으로 줄어드는 것을 확인 할 수 있다.

그림 28

(2) 평가

UNet은 Fully Convolutional Network(FCN)의 개념과 스킵 연결을 활용하여 이미지 분할 작업에 특화되어 있다. 그 결과, 상세한 특징을 추출하고 높은 정확도로 물체의 경계를 정확하게 분할할 수 있다. UNet의 높은 성능은 적은 데이터셋에서도 탁월한 성과를 보여줘서 데이터 수집에 어려움을 겪는 의료 영상 세그멘테이션 분야에 유용하게 활용되고 있기에 비슷한 어려움을 겪고 있는 본과제에서도 괜찮은 성능을 낼 것으로 기대된다. 따라서 UNet을 본 연구에 적합한 모델로 선택하였다.

3. 현재까지의 진행 상황 및 분석

3-1. 모델 구현

```
import os
import numpy as np

import torch
import torch.nn as nn

## 네트워크 구축하기
class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()

        def CBR2d(in_channels, out_channels, kernel_size=3, stride=1,
padding=1, bias=True):
            layers = []
            layers += [nn.Conv2d(in_channels=in_channels,
out_channels=out_channels,
                                 kernel_size=kernel_size, stride=stride,
padding=padding,
                                 bias=bias)]
            layers += [nn.BatchNorm2d(num_features=out_channels)]
            layers += [nn.ReLU()]

            cbr = nn.Sequential(*layers)

            return cbr

        # Contracting path
        self.enc1_1 = CBR2d(in_channels=1, out_channels=64)
        self.enc1_2 = CBR2d(in_channels=64, out_channels=64)
```

```
        self.pool1 = nn.MaxPool2d(kernel_size=2)

        self.enc2_1 = CBR2d(in_channels=64, out_channels=128)
        self.enc2_2 = CBR2d(in_channels=128, out_channels=128)

        self.pool2 = nn.MaxPool2d(kernel_size=2)

        self.enc3_1 = CBR2d(in_channels=128, out_channels=256)
        self.enc3_2 = CBR2d(in_channels=256, out_channels=256)

        self.pool3 = nn.MaxPool2d(kernel_size=2)

        self.enc4_1 = CBR2d(in_channels=256, out_channels=512)
        self.enc4_2 = CBR2d(in_channels=512, out_channels=512)

        self.pool4 = nn.MaxPool2d(kernel_size=2)

        self.enc5_1 = CBR2d(in_channels=512, out_channels=1024)

        # Expansive path
        self.dec5_1 = CBR2d(in_channels=1024, out_channels=512)

        self.unpool4 = nn.ConvTranspose2d(in_channels=512, out_channels=512,
                                          kernel_size=2, stride=2, padding=0,
bias=True)

        self.dec4_2 = CBR2d(in_channels=2 * 512, out_channels=512)
        self.dec4_1 = CBR2d(in_channels=512, out_channels=256)

        self.unpool3 = nn.ConvTranspose2d(in_channels=256, out_channels=256,
                                          kernel_size=2, stride=2, padding=0,
bias=True)

        self.dec3_2 = CBR2d(in_channels=2 * 256, out_channels=256)
        self.dec3_1 = CBR2d(in_channels=256, out_channels=128)

        self.unpool2 = nn.ConvTranspose2d(in_channels=128, out_channels=128,
                                          kernel_size=2, stride=2, padding=0,
bias=True)

        self.dec2_2 = CBR2d(in_channels=2 * 128, out_channels=128)
        self.dec2_1 = CBR2d(in_channels=128, out_channels=64)

        self.unpool1 = nn.ConvTranspose2d(in_channels=64, out_channels=64,
                                          kernel_size=2, stride=2, padding=0,
bias=True)

        self.dec1_2 = CBR2d(in_channels=2 * 64, out_channels=64)
        self.dec1_1 = CBR2d(in_channels=64, out_channels=64)

        self.fc = nn.Conv2d(in_channels=64, out_channels=1, kernel_size=1,
stride=1, padding=0, bias=True)
```

```
def forward(self, x):
    enc1_1 = self.enc1_1(x)
    enc1_2 = self.enc1_2(enc1_1)
    pool1 = self.pool1(enc1_2)

    enc2_1 = self.enc2_1(pool1)
    enc2_2 = self.enc2_2(enc2_1)
    pool2 = self.pool2(enc2_2)

    enc3_1 = self.enc3_1(pool2)
    enc3_2 = self.enc3_2(enc3_1)
    pool3 = self.pool3(enc3_2)

    enc4_1 = self.enc4_1(pool3)
    enc4_2 = self.enc4_2(enc4_1)
    pool4 = self.pool4(enc4_2)

    enc5_1 = self.enc5_1(pool4)

    dec5_1 = self.dec5_1(enc5_1)

    unpool4 = self.unpool4(dec5_1)
    cat4 = torch.cat((unpool4, enc4_2), dim=1)
    dec4_2 = self.dec4_2(cat4)
    dec4_1 = self.dec4_1(dec4_2)

    unpool3 = self.unpool3(dec4_1)
    cat3 = torch.cat((unpool3, enc3_2), dim=1)
    dec3_2 = self.dec3_2(cat3)
    dec3_1 = self.dec3_1(dec3_2)

    unpool2 = self.unpool2(dec3_1)
    cat2 = torch.cat((unpool2, enc2_2), dim=1)
    dec2_2 = self.dec2_2(cat2)
    dec2_1 = self.dec2_1(dec2_2)

    unpool1 = self.unpool1(dec2_1)
    cat1 = torch.cat((unpool1, enc1_2), dim=1)
    dec1_2 = self.dec1_2(cat1)
    dec1_1 = self.dec1_1(dec1_2)

    x = self.fc(dec1_1)

    return x
```

3-2. 분석

변경된 목표 – 대형관에서 용접선 찾기를 위한 적절한 데이터 전처리를 시도해 보았다. 하지만 데이터 전처리가 효과적이라 판단되지 않아 전처리는 최소화하여 모델을 개발하기로 하였다. 또한, 착수보고서에서 언급한 YOLO 모델이 적절하지 않다고 판단하여 다른 여러 딥러닝 모델 후보들을 조사하였다, 최종 후보인 ResNet과 UNet 중 UNet이 가장 적합한 모델이라 판단하여 UNet을 사용하기로 하였다. 왜냐하면 UNet 모델은 인코더-디코더 구조와 스킵 연결을 활용하여 높은 정확도의 이미지 분할을 수행하며, 적은 데이터로도 우수한 성능을 보여주기 때문이다. 그 후에 해당 모델의 코드를 구현도 해 보았다.

현재까지의 분석 결과, 대형관 용접선 위치를 탐지하는 모델을 개발하는 것이 가능할 것 같다. UNet 모델은 이미지 분할 작업에 특화되어 있기 때문에, 꽤 괜찮은 정확도로 용접선의 위치를 찾을 수 있을 것으로 기대된다. 이 모델을 통해 사람이 직접 검사하는 것보다 빠르고 정확하게 용접선의 위치를 찾을 수 있게 되며, 향후 결함 탐지 모델 개발에도 이 모델을 활용하여 성능을 더욱 향상할 수 있을 거라 기대된다.

4. 보완점과 앞으로의 진행 방향

현재 연구에서는 데이터의 양이 적기 때문에 데이터 증강 기법을 활용하여 모델의 성능을 향상시킬 필요가 있다. OpenCV와 같은 라이브러리를 사용하여 이미지를 회전, 뒤집기, 확대/축소 등의 변형을 주어 데이터를 다양하게 한다면, 좀 더 정확도 있는 모델을 얻을 수 있을 거라 생각한다. 가능하다면 추가 자료를 받아오는 것도 좋을 것이다.

또한, UNet 모델의 파라미터를 최적화하여 모델의 성능을 개선해야 할 것이다. 적절한 학습률, 배치 크기, 학습 수 등을 설정하여 학습 과정을 최적화하고, 모델의 아키텍처에 변경을 가하여 더 높은 정확도와 효율성을 달성하는 것을 목표로 삼아야 할 것이다.

또한 가능하다면 모델을 배포할 수 있는 적절한 매체를 만들어 쉽게 모델을 사용할 수 있게 만들 것도 고려하고 있다. 노트북 등을 이용해 간단한 서버를 만들

어 그곳에 접속하여 쉽게 모델을 사용하게 한다면 편리성 면에서 좀 더 개선할 수 있을 것이다.

만약에 모델이 기대 이상의 성능을 보여주고, 개발 진척 속도가 빠르다면 추가로 초기 연구 목적이었던 결함 탐지 모델을 개발하는 시도를 시작할 수도 있을 것이다. 아니면 다른 딥러닝 모델을 사용하여 성능 비교를 통해 좀 더 우수한 모델을 선별해 내고 더 좋은 결괏값을 가지게 할 수도 있을 것이다.

5. 구성원별 진척도

이름	진척도
지민철	데이터 전처리, 딥러닝 모델 개발, 보고서 작성
이현규	딥러닝 모델 선정 및 개발, 모델 학습, 보고서 작성
정다현	딥러닝 모델 선정, 모델 검증, 보고서 종합