2022 후기
졸업과제
중간보고서

철판 결함 검출을 위한 딥러닝 기법 설계

분과명 및 팀번호 : A, 2조
팀명 : 졸업시켜조
지도교수명 : 감진규 교수님

201824633 김유진
201924647 김지훈
201624472 문정민

목차

1. 기존 계획 및 추가 요구사항

(1) 기존 목표

최근 철판 관련 산업이 부상함에 따라 고품질 철판에 대한 수요가 상승하고 있다. 그러나 철판을 제조하는 과정에서 결함은 필연적이다. 철판의 원료나 철판을 제조하는 과정인 압연공정, 시스템 등의 이유로 흠집, 흉터, 그을림 등의 결함이 발생한다. 이 중 일부 결함이 발생했을 때 철판 품질에 치명적인 영향을 끼칠 수 있다. 현재 철판 검사는 보통 수동으로 이루어지고 있기 때문에 시간 및 비용적인 측면에서 매우 비효율적이다. 따라서 Deep Learning을 이용해 자동으로 결함을 식별하는 방법을 필요로한다.

본 과제는 Deep Learning을 이용해 철판 결함을 식별하고, 분류하는 알고리즘 모델을 설계하는 것이 목표이다. 학습을 위해 이미지 데이터를 전처리하는 방법에 따라 정확도와 손실을 비교 및 피드백한다. 최적으로 전처리된 이미지를 사용해 CNN 모델을 바탕으로 Detection 및 Classification의 정확도와 손실을 비교하고 피드백한다. 학습된 모델을 적용해 결과를 GUI 형태로 나타낸다.

(2) 추가 요구사항 및 대책

a. 데이터 전처리

기존에는 이미지 데이터의 전처리를 수행할 때 Resize, Zoom In/Out, 수평/수직 반전 기능을 사용했다. 그러나 이 프로젝트에서 사용한 Dataset의 철판 이미지의 경우 너비가 높이에 비해 훨씬 큰 형태를 가지고 있어 Resize 시 데이터가 손실되어 모델 학습에 문제가 되었다. 이에 따라 Resize 대신 결함 부분을 Crop 하는 형식으로 변경했다. Zoom Out 필터는 축소되면 padding 형태가 나타나는데 이 부분이 굳이 학습에 도움이 되지 않는다고 판단하여 제거했다.

기존 요구사항	Image Augmentation을 이용한 데이터 전처리 - Resizing 256 x 256 - Rescailing 1./255 - Random Zoom In/Out x 0.1 - Random Horizontal Flip - Random Vertical Flip
수정 및 추가 요구사항	이미지 Resizing을 256 x 256에서 학습 모델의 input에 맞추어 224 x 224로 이미지를 crop 했다. Data Augmentation에서 Zoom Out을 제외했다.. 추후 밝기 등의 Augmentation Filter들을 적용해볼 예정이다.

b. 철판 결함 검출을 위한 학습 모델

기존에는 ResNet50의 Skip Connection을 활용하기 위해서 모델을 직관적으로 보기 위해 한 단계씩 쌓아 올렸다. 그러나 Skip Connection을 위한 모듈을 사용하는 것이 더 좋으리라 판단해 해당 모듈을 이용할 수 있도록 코드를 수정했다.

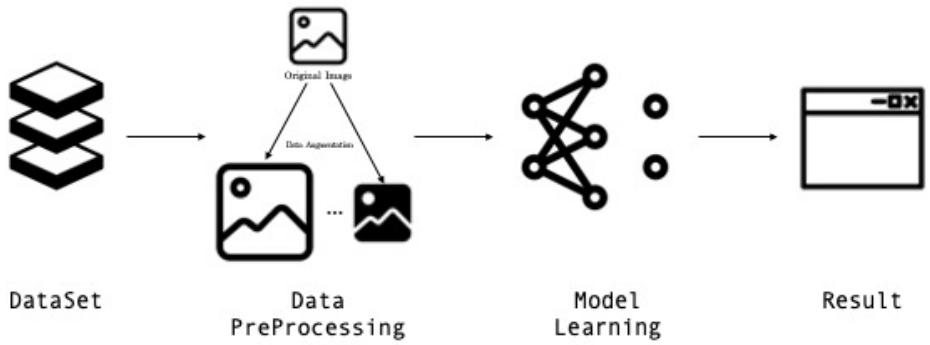
기존 요구사항	기존 모델을 토대로 개선된 모델의 제작을 진행 - Detection을 위한 Xception Model 환경 설정 - Classification을 위한 Resnet50 Model 환경 설정 - Resnet50의 Skip Connection을 이용해 모델의 정확도를 높이는 방식으로 진행해 개선된 데이터 모델 도출 - 간단한 테스트를 통해 매개변수 및 모델 층 조정
수정 및 추가 요구사항	ResNet50 모델을 직접 쌓아올리는 것이 아니라 모듈을 이용해 Skip Connection을 활용한다.

c. 결과 시각화

학습된 모델이 존재하지 않아 우선 형식만 만들었다. tkinter를 이용하려고 했으나 시각적인 효과를 위해 pyqt 모듈로 변경했다.

기존 요구사항	학습된 모델을 활용해 결과의 시각화 - tkinter를 이용한 GUI 설계
수정 및 추가 요구사항	tkinter를 이용해 GUI를 설계하는 것이 아닌, pyqt를 이용했다. 원본 이미지와 전처리된 이미지를 나타내 한눈에 비교할 수 있도록 한다.

2. 세부 프로그램 설계 명세 및 계획



<그림 1.> 전체적인 pipeline

(1) 데이터 선정

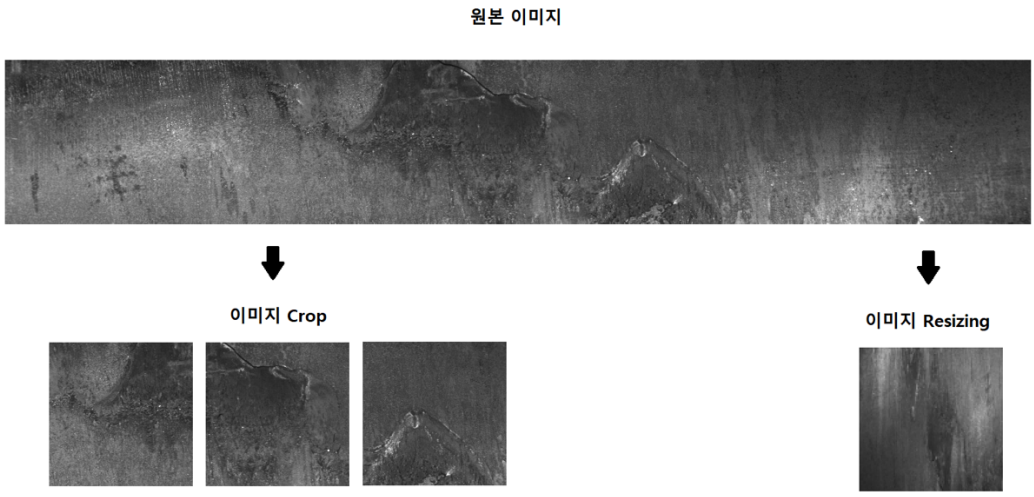
<그림 2.> Severstal 데이터셋의 예시 이미지

이 프로젝트에서는 Kaggle의 Severstal Dataset을 이용한다. 결함 4개에 대해 총 12,568개의 이미지가 존재하며 1600 x 256 크기의 이미지들로 구성되어있다. 픽셀 단위로 결

함이 제공되기 때문에 Segmentation을 활용해 학습 모델 생성도 가능하고, Object Detection을 이용도 가능해 활용성이 높기 때문에 이 Dataset을 선정하게 되었다.

(2) 데이터 전처리

Dataset의 unbalancing 문제와 정확도 향상을 위해 Image Augmentation 기법을 사용했다. 이미지 전처리를 위해 Crop 및 Rescaling 후 무작위 수평/수직 반전 기능을 사용했다.



<그림 3.> 이미지 Crop 및 Resizing 비교 예시

이미지 전처리에서 가장 먼저 Crop을 진행했다. 이미지의 특정 부분을 잘라내어 학습에 사용하는 Crop의 경우 Resizing과 다르게 데이터의 손실이 거의 없어 모델 학습 시 가장 이상적인 방법이라고 생각되어 사용했다. 아직 테스트를 위한 환경이 구축되지 않아 추후 Resize를 통한 정확도와 Crop을 통한 정확도를 비교해볼 예정이다. 이 프로젝트에서 사용할 모델인 ResNet50은 224 x 224 형태의 이미지에 대한 학습을 지원하므로 이 크기에 맞추어 Crop을 진행했다.

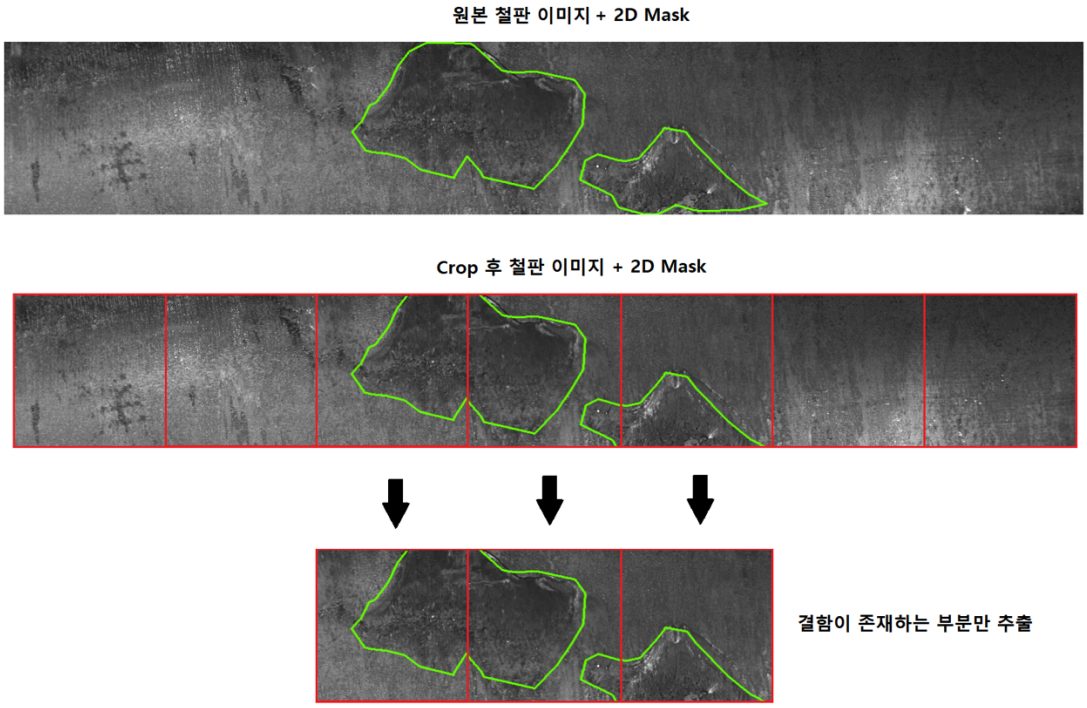
```
SHAPE = (1600, 256) # 원본 이미지 shape (너비 * 높이)
TARGET_SHAPE = 224 # 크롭 이미지의 shape
CROPS = 7 # 하나의 이미지에서 생성될 크롭될 이미지의 개수
OFFSETS = [16 + TARGET_SHAPE*i for i in range(CROPS)] # 원본 이미지에서 크롭할 위치에 대한 오프셋
```

기존 철판 이미지는 1600 x 256 형태이므로 224 픽셀 단위로 자르면 총 7개의 Crop 이미지 생성이 가능하다.

```
# 이미지 크롭을 수행하고 결과를 zip 파일로 출력
def crop_images(train_df):
    # 크롭된 이미지들을 zip 파일로 출력
    with zipfile.ZipFile(CROPPED_IMGS_ZIP, 'w') as cropped_imgs_out:
        # csv 데이터 읽기 & 이미지 불러오기
        for imageId, classId, encodedPixels in tqdm(train_df.values):
            img = Image.open(TRAIN_IMG_DIR + "{}".format(imageId))
```

```
            img = np.asarray(img)
            mask = encodedPixelsTo2DMask(encodedPixels)
            # 이미지 크롭 수행, 크롭해서 생성할 이미지의 shape를 토대로 정해진 수만큼 원본 이미지를 크롭
            # 크롭된 이미지에 결함 부분이 존재하지 않는 경우 제외, 존재하는 경우 유지한다
            # 결함의 크기가 큰 경우 해당 결함에 대해서 여러 개의 크롭된 이미지가 존재할 수 있음
            for i in range(CROPS):
                cropped_img = img[16:-16, OFFSETS[i]:OFFSETS[i]+TARGET_SHAPE, :1]
                cropped_mask = mask[16:-16, OFFSETS[i]:OFFSETS[i]+TARGET_SHAPE]
                # 크롭된 이미지에 결함 부분이 포함되지 않은 경우
                if cropped_mask.max() == 0:
                    pass
                # 크롭된 이미지에 결함 부분이 포함된 경우
                # 크롭된 이미지 파일명 = 이미지 이름_결함 클래스_크롭 순번.png
                else:
                    imgName = imageId[:-4] + '_' + str(classId) + '_' + str(i) + '.png'
                    cropped_img = cv2.imencode('.png', cropped_img)[1]
                    cropped_imgs_out.writestr(imgName, cropped_img)
```

이미지를 Crop 한 뒤 csv 파일의 EncodedPixels를 기존 철판 이미지의 결함 위치를 표시하는 역할의 2D Mask로 변환한다. 그 후 Crop 된 이미지와 대조해 결함 부분이 있는 Crop 이미지를 저장한다. 만들어진 Crop 이미지들은 zip 파일 형식으로 저장한다.



<그림 4.> 원본 이미지에 Crop을 수행한 결과, 이 경우 3개의 Crop 이미지가 생성

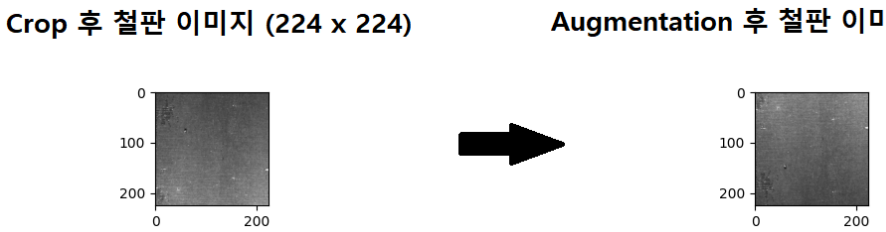
만약 하나의 결함이 이미지의 상당 부분을 차지하는 경우, 한 결함에 대해 여러 개의 Crop된 이미지가 생성될 수 있다. 이 경우 학습 시 무작위로 하나를 선택하는 방식을 사용해 데이터의 개수를 늘리는 효과를 볼 수 있다. 그 후 Rescaling을 통해 픽셀의

Intensity를 0~255에서 0~1 범위로 조정한다.

```
def augmentation(imgs):
    # 이미지 데이터에 적용시킬 augmentation 기법 설정
    generator = tf.keras.Sequential([
        pp.Rescaling(1./255),
        pp.RandomFlip('horizontal'),
        pp.RandomFlip('vertical')
    ])

    flag = 0
    for it in tqdm(batch(imgs)):
        if flag != 0:
            it = generator(it)
            augmented_imgs = np.vstack((augmented_imgs, it))
        else:
            augmented_imgs = generator(it)
            flag = 1

    return augmented_imgs
```



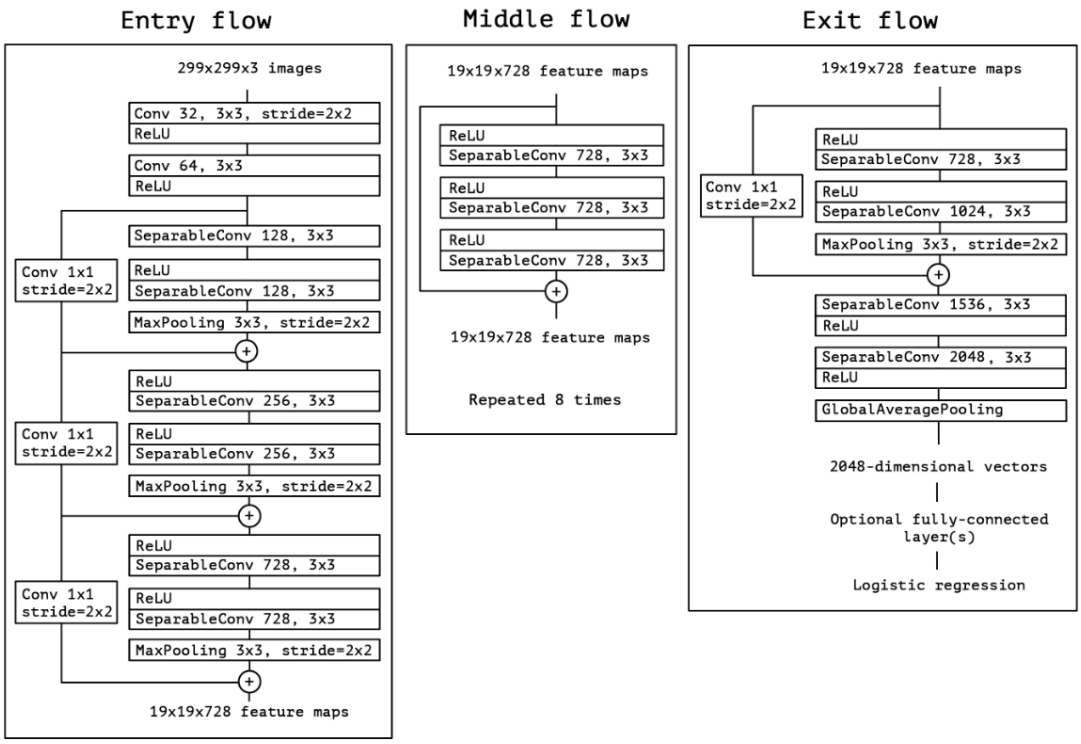
<그림 5.> 무작위 수평/수직 반전 결과

데이터 전처리 이후 학습에 사용할 데이터 개수를 늘리기 위해 Data Augmentation 기법을 사용한다. 이미지 데이터에서 Data Augmentation 기법은 원본 이미지의 특성을 유지하면서 약간의 수정을 거쳐 새로운 이미지를 만들어낸다. 우선적으로 무작위 수평/수직 반전 필터를 적용했다.

이후 다양한 Augmentation 필터를 적용해보고 결과를 비교해 최선의 방식을 찾아갈 예정이다.

(3) 대표 모델 테스트

Xception Model로 결함 유무를 판별한 뒤, 결함이 있는 이미지만 Resnet50 모델을 통해 Classification하는 방법을 이용했다.



<그림 6.> Xception의 구조 : Entry Flow -> Middle Flow (8회 반복) -> Exit Flow

Input으로 원본 이미지를 그대로 넣어주려고 했다. Xception의 경우 고화질의 이미지에 적합한 학습 모델이다. 일반적으로 고화질 이미지라고 하면 HD 해상도를 이야기한다. 이 프로젝트에서 사용하는 DataSet의 이미지 크기는 1600 x 256으로 HD 해상도 픽셀 수의 반 정도이다. 이에 따라 Xception 모델을 이용해 결함의 유무를 Detection 하는 방법을 이용했다.

```
base_model =
tf.keras.applications.Xception(input_shape=(1600,256,3),include_top=False,weights="imagenet
")

for layer in base_model.layers[:-5]:
    layer.trainable=False

model=Sequential()
model.add(base_model)
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(256,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(256,activation="relu"))
model.add(Dense(1,activation="softmax"))
```

Input으로 1600 x 256 크기의 이미지가 들어오고 binary 형태의 ClassId가 들어온다. 결함의 유무만 판별해주면 되기 때문에 train 데이터를 불러올 때 class의 mode를 binary로 설정했다. 이에 따라 출력층의 뉴런 수를 1개로 설정했다. Exit Flow 부분을 False로

설정해준 뒤 output에 맞춰 모델을 쌓아 올렸다. 이 모델을 이용해 ClassId가 1로 판별된 이미지들을 Resnet50의 Input으로 넣는다.

ResNet50은 50개의 층과 2300만 개의 파라미터로 이루어진 CNN 구조의 모델이다.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

<그림 7.> Resnet50의 구조

ResNet은 Skip Connection을 이용해 성능이 향상된 모델이다. 이를 이용해 Classification을 진행했을 때 좋은 결과가 나타났다는 선례 연구를 참조해 해당 모델을 사용하게 되었다. 또한 imagenet DataSet을 활용해 weights를 통해 간단한 테스트도 가능하다.

```
base_model = ResNet50(include_top=True, input_shape = (224, 224 ,3), weights = 'imagenet')
```

이미지 데이터 전처리를 통해 얻은 데이터 크기인 224 x 224를 모델에 적용했다. 아직 제대로 된 학습을 시도하지 않았기 때문에 weights 매개변수에 ImageNet을 부여했다.

현재까지는 Object Detection 후 Classification 하는 방법을 활용했다. Detection을 위해 고화질 이미지에 적합한 학습 모델인 Xception 모델을 이용했으나 DataSet의 특성상 결함이 없는 이미지가 존재하지 않는다. 뿐만 아니라 224 x 224의 전처리된 이미지를 받게되면, 고화질의 이미지를 처리할 필요가 없기 때문에 아래의 두가지 방법을 테스트해 볼 예정이다. 첫 번째는 Resnet50만을 이용해 Detection 단계를 없애는 방식이다. ClassId 1 ~ 4번과 더불어 결함 없음(0번)을 추가한다. 두 번째는 Input image size를 수정해 결함이 없는 이미지 DataSet을 확보한 뒤 기존방법을 활용하는 방식이다. 실험 후 더 좋은 방식을 채택할 예정이다. 그뿐만 아니라 Resnet50의 Skip Connection Module의 매개변수 조정을 통해 성능을 비교해볼 예정이다.

한 이미지에 여러 개의 결함이 있는 DataSet의 특성상 Segmentation 기법을 활용한다면 정확도가 더 높아질 것으로 예상한다. 이에 따라 UNet 모델을 적용해볼 예정이다. UNet은 적은 양의 데이터로도 비교적 정확한 Segmentation을 해낸다는 특징이 있다. 이 프로젝트에서 사용하는 DataSet의 경우 특정 클래스에 데이터가 치우쳐있으므로 이 모델을 적용했을 때 더욱 효과적일 것으로 보인다.

(4) 결과 시각화

프로젝트의 결과를 쉽게 눈으로 보기 위한 방법을 모색하다가, 모델 구축 및 이미지

전처리를 모두 Python으로 진행하기 때문에 Python의 모듈을 이용한 GUI 형태로 나타내기로 했다. 사용자가 편리하게 프로그램을 이용하는 것이 GUI의 가장 큰 역할이라고 생각해 단순히 프로그램 실행을 통해 결과만 확인하는 것이 아니라 원본 이미지와 전처리 이후 이미지를 한눈에 비교해 어떤 결함이 존재하는지 파악해야 한다. 아직 모델을 적용해보지 않아 임의로 각 결함 DataSet의 개수 현황을 그래프로 나타냈다. 이때 사용한 모듈은 pyqt이다.

<그림 8.> Main 화면

파일 불러오기 버튼을 이용해 폴더를 선택 후 실행 버튼을 클릭하면 SteelDefection.py 파일이 실행된다.

<그림 9.> 실행 화면

이미지 목록에는 폴더 내부의 이미지들의 목록이 나타나게 되고, 리스트에 있는 이미지를 클릭할 때마다 전처리 전 이미지가 실제 이미지 칸에 나타나고, 전처리 후 이미지가 아래 칸에 나타난다. 결과 화면에는 우선 class별 결함의 개수를 그래프로 나타냈다.

새로운 이미지에 대해서 학습 모델을 활용하여 철판의 결함이 존재 여부와 만약 존재한다면 결함의 종류가 무엇인지를 화면에 나타낼 예정이다. 더 나아가 해당 화면에서 전처리 및 모델을 학습하는 방법도 고려할 것이다.

3. 갱신된 과제 추진 계획 및 진척도

(1) 갱신된 과제 추진 계획

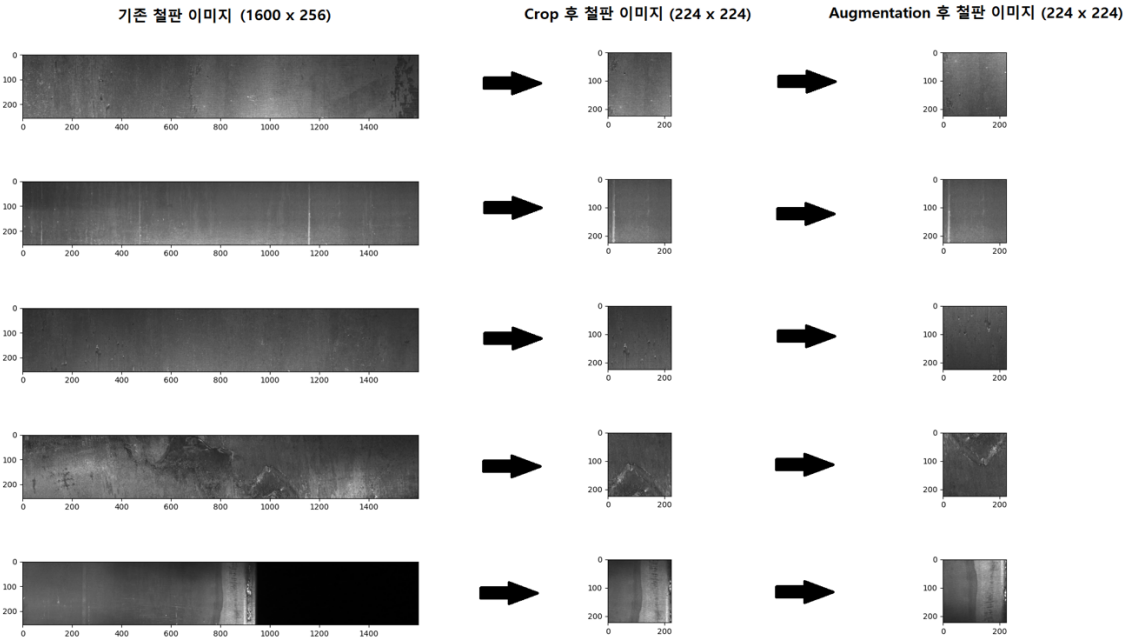
(2) 진척도

이름	진척도
김유진	개발 환경 설정 및 기존 논문 연구 - 모델 학습을 위해 DataSet의 구성 연구 - 모델 학습을 위한 환경 구성 Detection Model 선정 - Xception을 통한 Detection - (input) 1600 x 256 size - (output) Binary (1, None) Classification Model 선정 - ResNet50을 이용한 Classification - (input) 전처리된 이미지 224 x 224 size - (output) Class 별 유사도 (4, None)
김지훈	개발 환경 설정 및 기존 논문 연구 - 결과 처리를 위해 pipeline의 모든 output들 검토 - 전반적인 프로젝트 흐름 검토 - GUI를 위한 환경 구성 GUI 전체적인 틀을 짜고 세부적인 연결 작업(이미지 매핑) 진행중
문정민	개발 환경 설정 및 기존 논문 연구 - 이미지 전처리를 위해 DataSet의 구성 연구 - 데이터 전처리를 위한 환경 구성 이미지 데이터 전처리 및 Augmentation - Crop 224 x 224 - Rescaling 1./255

	- Random Horizontal / Vertical Flip

4. 현시점까지의 중간 결과

(1) 이미지 전처리



<그림 10.> 이미지 전처리 중간 결과

기존 1600 x 256 형태의 철판 이미지를 224 x 224의 결함 부분을 나타내는 이미지로 Crop하고, 이를 랜덤 수평/수직 반전을 거쳐 학습에 사용될 이미지를 만든다.

(2) 모델 테스트

Colab 사용 시간 이슈로 인해 작은 횟수의 학습을 진행했다. 아래 그래프는 Xception Model 학습의 결과이다.

<그림 11.> Epoch : 30, batch_size : 16 일때, 각 모델에 대한 loss 와 accuracy

(3) 결과 시각화

<그림 12.> 현재까지의 GUI 시각화 결과

5. 출처 및 참조

a. Kaggle Severstal: Steel Defect Detection. Can You Detect and Classify Defects in Steel? 2019. Available online: https://kaggle.com/c/severstal-steel-defect-detection

b. Steel Surface Defect Classification Using Deep Residual Neural Network. 26 June 2020.