# 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

# 1. 설계 상세화 및 변경 내역

## 1.1 개발환경

- 개발 언어 : Python3
- 개발 도구 : Google Colab, tensorflow 등
- 실행 환경 : Window, Mac

## 1.2 참고 자료 (논문)

- Metal Defect Classification Using Deep Learning
- Mixed supervision for surface-defect detection
- Steel Surface Defect Detection using Deep Learning_main refererence
- TLU-Net A Deep Learning Approach for Automatic Steel Surface Defect Detection

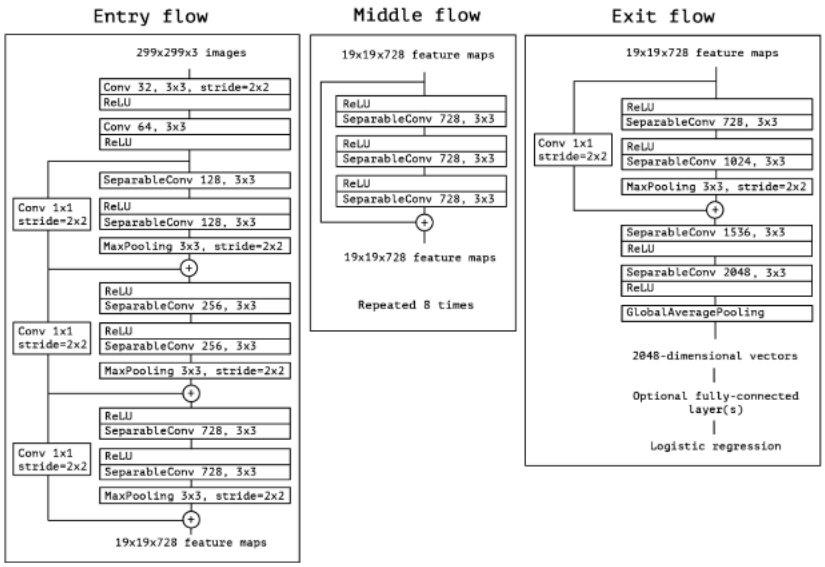
## 1.3. 사용 기술

[이미지 전처리] Image Augmentation

[프로그램 시각화] Python 의 GUI module

[이미지 학습]

- Xception




<그림 1.> Xception Flow Chart

Xception 은 Modified Depthwise Separable Convolution 이라고 할 수 있다. 기존 Depthwise Separable Convolution 과 다르게 pointwise 와 depthwise 의 순서가 변했고, ReLU 가 존재하지

않는다. 뿐만 아니라, Residual connection 이 존재함으로써 정확도가 높아졌다. 이 기술은 연산량과 parameter 의 개수를 줄여서, 큰 이미지 인식을 고속화 시키는 것이 목적이다.

Xception 은 총 14 개 모듈로 이루어져있고, 총 36 개의 convolutional layer 가 존재한다. feature map 은 entry, middle 그리고 exit 까지 3 가지로 구성되어 있으며, 입력값은 Entry flow 로 들어가서 midlle flow 를 8 번 거친 뒤 exit flow 를 통과해 이미지를 분류한다.

- Resnet50

Resnet50 은 50 개의 layer 로 구성된 CNN 구조의 model 이다.

| layer name | output size | 18-layer | 34-layer | 50-layer |
|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | |
| conv2_x | 56×56 | 3×3 max pool, stride<br>$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| | 1×1 | average pool, 1000-d fc, s | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ |

<표 1.> Resnet50 의 구조 50-Layer 부분

Input 의 크기는 224 x 224 이고, Detection 보다는 이미지를 분류할때 주로 사용한다.

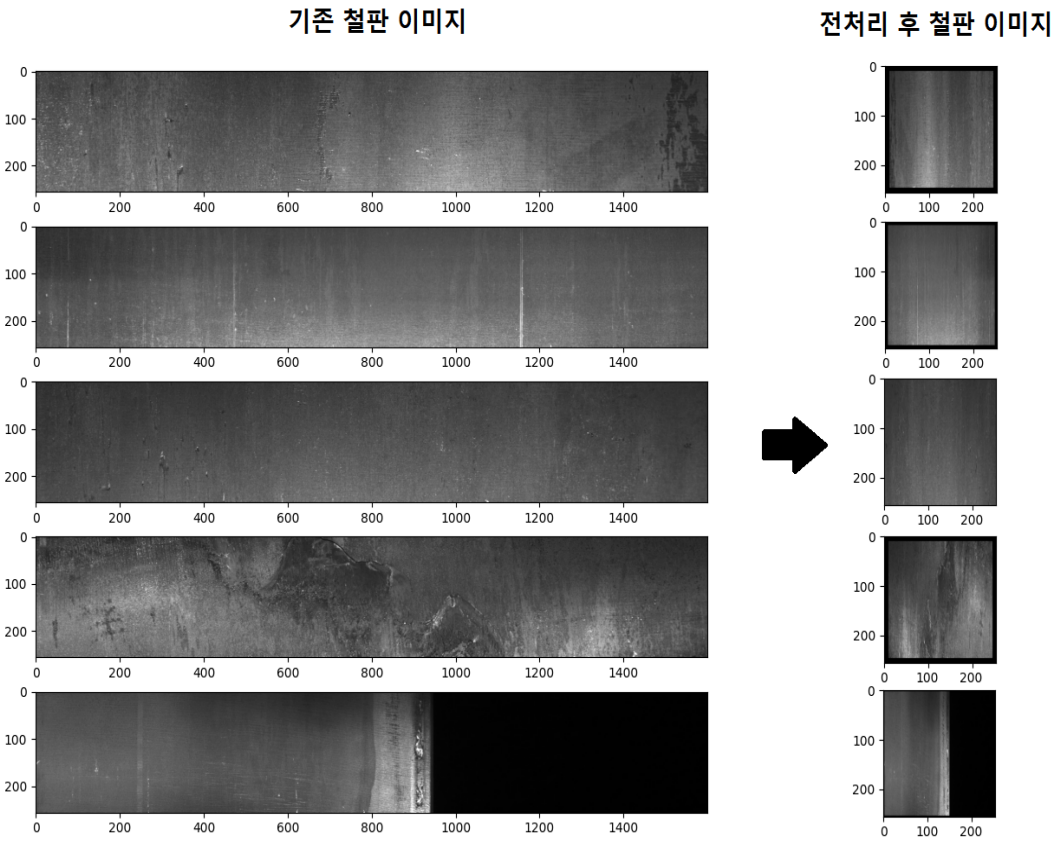
# 2. 보고 시점까지의 과제 수행 내용 및 중간 결과

공통 범위였던 논문 읽기 및 Kaggle 의 Dataset 을 조사하고 이해했다.

## 2.1 데이터 전처리

```
generator = tf.keras.Sequential([
        pp.Resizing(255, 255),
        pp.Rescaling(1./255),
        pp.RandomZoom(0.1, fill_mode='constant', fill_value=0),
        pp.RandomFlip('horizontal'),
        pp.RandomFlip('vertical')
])
```

Image augmentation 을 위한 코드는 위와 같다. 이미지의 크기를 256 x 256 으로 변형한 뒤, Intensity 값의 범위를 0~255 에서 0~1 로 변경했다. 이후 무작위로 좌우 반전을 시키거나, 확대하거나 축소시키는 방식을 적용했다. 이에 따른 이미지의 예시는 다음과 같다.




<그림 2.> Image Augmentation 결과

현시점까지 데이터 전처리를 통한 성능 확인을 진행하지 못해, 먼저 틀 구축만을 해뒀다. 이후 실제 데이터 이미지에 맞도록 크기를 변경하고, 더 많은 augmentation 필터를 이용해 성능 향상에 이바지할 것이다. 그뿐만 아니라, 각 클래스에 대한 이미지 수가 균등하도록 augmentation 하는 방법을 이용할 예정이다.

## 2.2 대표 모델 테스트

Binary Detection 을 통해서 결함의 유무를 판별한 뒤, Classification 하는 방향으로 Base Model 을 구축했다. 사용한 모델은 Xception 을 이용하는 방법을 적용했고, 이를 통해 존재한다고 판단되는 이미지의 결함을 Resnet50 을 이용해 class 에 맞게 분류한다. Resnet50 을 베이스로 수정해나가는 것으로 계획했기 때문에, 모듈이 아닌 직접 Model 을 쌓아올렸다.

a. Xception

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Functional)        (None, 4, 4, 2048)        20861480

dropout (Dropout)            (None, 4, 4, 2048)        0

flatten (Flatten)            (None, 32768)             0

dense (Dense)                (None, 256)               8388864

dropout_1 (Dropout)          (None, 256)               0

dense_1 (Dense)              (None, 128)               32896

dropout_2 (Dropout)          (None, 128)               0

dense_2 (Dense)              (None, 256)               33024

dense_3 (Dense)              (None, 4)                 1028

=================================================================
Total params: 29,317,292
Trainable params: 11,622,532
Non-trainable params: 17,694,760
_________________________________________________________________
```

Input 은 128 x 128 사이즈로 적용했으나, 추후 전처리를 통해 받을 데이터의 사이즈에 따라 수정예정이다. Output 은 클래스의 개수와 사이즈가 동일하다.

```
base_model = tf.keras.applications.Xception(input_shape=(128,128,3),
    include_top=False, weights="imagenet")

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(4,activation="softmax"))
```


<그림 3.> Epoch : 30, batch_size : 16 일때, 각 모델에 대한 loss 와 accuracy

Colab의 이용 시간에 쫓겨 30회의 epoch만을 테스트해볼 수 있었다. 논문을 참고해 epoch나 batch_size를 조정해 테스트해볼 계획이다.

b. Resnet50

전이학습의 개념을 몰라 Resnet50 의 구조를 분석해 레이어를 하나하나 쌓아올리는데 시간을 소요했다.

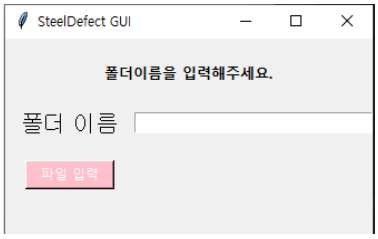
```
predicted_classes = []

for image_name in image_list:
    image_path = os.path.join('test_images', image_name)
    image = tf.keras.preprocessing.image.load_img(image_path, target_size=(128
, 128))
    image = tf.keras.preprocessing.image.img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = tf.keras.applications.resnet50.preprocess_input(image)
    predictions = model.predict(image)
    predicted_class = np.argmax(predictions, axis=1)
    predicted_classes.append(predicted_class)
```

위의 코드에 전이학습의 개념을 적용하여 추후 레이어를 덧붙이거나 제거하는 등의 작업을 이용해 성능을 개선해 나갈 것이다.

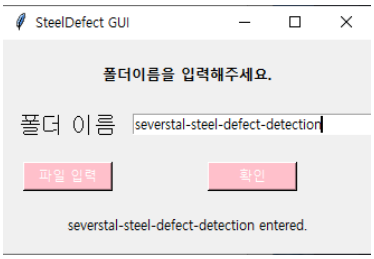
## 2.3 GUI 를 이용한 시각화 < SteelGUI >

현재 상태에서 할 수 있는게 없어, 우선적으로 data 의 개수에 따른 그래프를 GUI 를 통해 구현했다.

[메인화면]

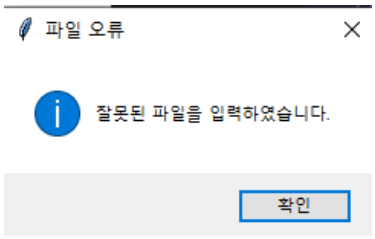


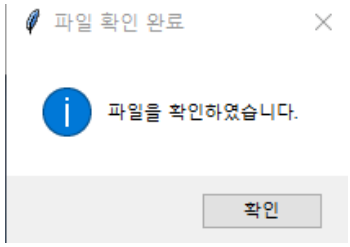
a. 분석 이미지가 들어있는 폴더명을 입력란에 입력하고 파일입력 버튼을 누른다.




b. 입력된 폴더명이 나타난다.

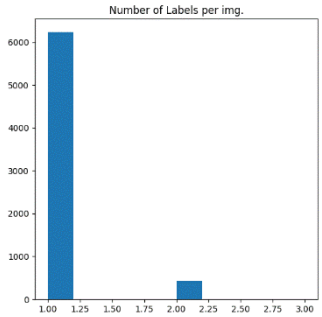
c. '확인' 버튼을 눌러 폴더가 있는지 확인한다.



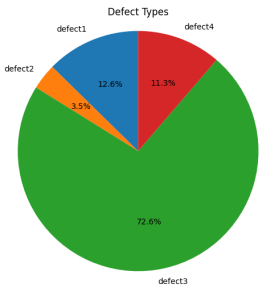




<잘못된 폴더가 입력된 경우>　　<올바른 폴더가 입력된 경우>

d. 올바른 폴더가 입력된 경우 '실행하기' 버튼이 생성되며, 버튼을 누르면 SteelDefection.py 파일이 실행되며 결과가 출력된다.




e. 결과 화면은 다음과 같다.



# 3. 갱신된 과제 추진 계획

a. 개발 프로세스 일정


b. 구성원별 진척도

| 구분 | 이름 | 담당 업무 |
|---|---|---|
| 조장 | 김유진 | Model 구축 및 테스트<br>-Xception을 이용한 Detection 환경 구축<br>-Resnet50을 이용한 Classification 환경 구축 |
| 조원 | 문정민 | Image Augmentation을 이용한 데이터 전처리 |
| 조원 | 김지훈 | GUI 형태로 시각화<br>-Number Of Labels Per Image |

# 4. 요구조건 및 제약 사항 분석에 대한 수정사항

## 4.1 기존 목표

본 과제에서는 Kaggle 에 공개된 Severstal: Steel Defect Detection 의 데이터를 전처리한 뒤, 학습시킨 모델을 통해 철판 표면의 결함을 검출해내고 분류한다. 이후 python GUI Tool 을 이용하여 이미지에 적용하는 것이 목표이다.

## 4.2 수정 사항 및 대책

[이미지 전처리] 기존과 목표는 동일하고 다양한 augmentation filter 를 사용해 결괏값을 도출하고, balancing 을 맞춰주는 작업이 필요하다.

[모델 구축] 전이 학습에 대한 이해도가 필요하고, 이에 대한 응용법을 찾아가고자 한다. 그뿐만 아니라, data segmentation 을 이용한 모델 구축에도 관심을 기울이고 있다.

[GUI] 앞선 전처리와 모델 구축에 병행해서 할 수 있는 작업이 아니다 보니, 해낼 수 있는 과제에 대한 한계가 존재했다. 학습된 모델과 test image 를 통해 결과를 그래픽 형태로 나타낼 수 있도록 해야 한다.