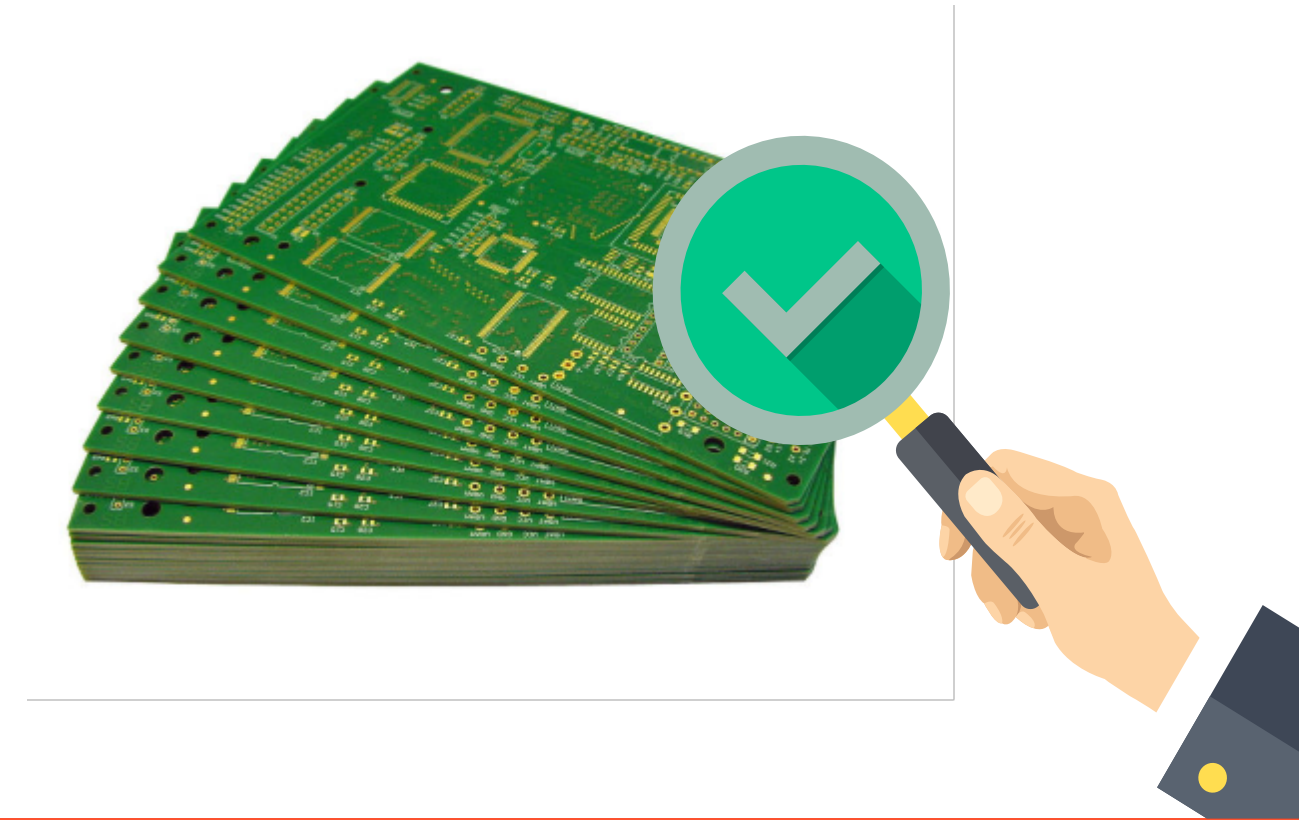


# PCB 불량검출

Deep Learning with Tensorflow

# CONTENTS

PCB 불량 검출



INTRODUCTION



모델 구현



결과



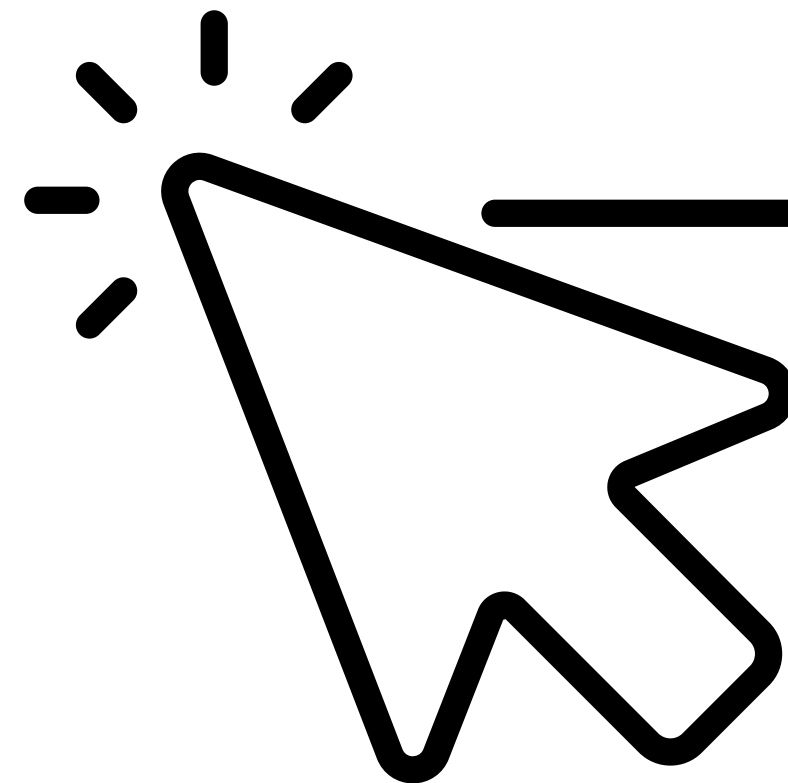
REFERENCES

01 |

Introduction

### PURPOSE OF PROJECT

PCB 이미지 데이터를 이용하여 불량 PCB 를 검출하는  
딥러닝 알고리즘 개발

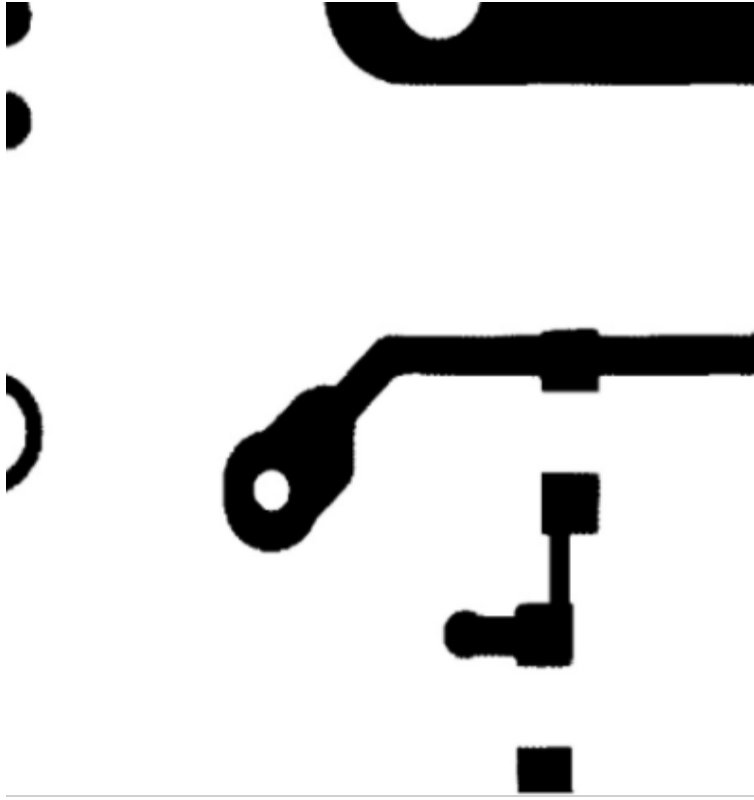
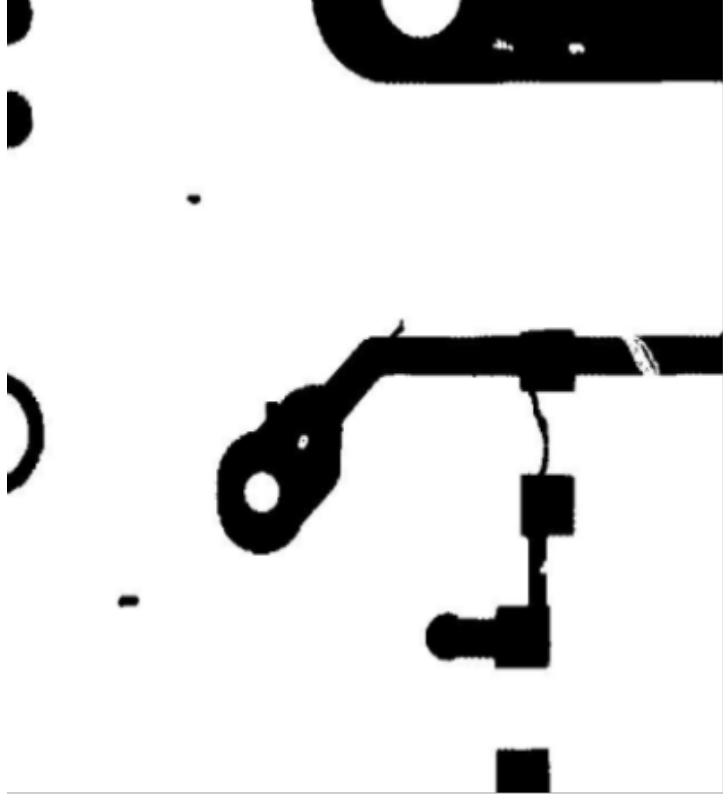


# 01 Introduction

## Kaggle Dataset

### Feature

#### 01 Kaggle Dataset

Normal PCB Image	Defect PCB Image
	

원활한 분할을 위해 비정상적인 이미지만 사용한다.

# Kaggle Dataset

Label

Defect PCB Log	데이터 세부설명	
	공백 기준 1 ~ 4열	공백 기준 5열
466 441 493 470 3 454 300 493 396 2 331 248 364 283 4 221 314 253 350 4 151 149 182 175 5 492 28 525 55 6 424 24 461 53 6 250 341 278 370 6 539 259 592 316 1 89 469 127 497 5	x1, y1, x2, y2 좌표값	1 : open 2 : short 3 : mousebite 4 : spur 5 : copper 6 : pinhole

# 01 Introduction

## Kaggle Dataset

### 01 Kaggle Dataset

#### Data Augmentation 적용

좌우 반전	상하 반전	핵심 파이썬 코드
		<pre>inputs = keras.layers.Input((128,128,1)) x = keras.layers.experimental.preprocessing.Rescaling(1./255)(inputs) x = keras.layers.experimental.preprocessing.RandomFlip(mode='horizontal_and_vertical')(x)</pre>

overfitting을 방지하고 성능 향상을 위해 적용한다.

# 02 | 모델 구현



# 02 모델 구현

# Modeling

## 모델 설계

### 01 모델 설계

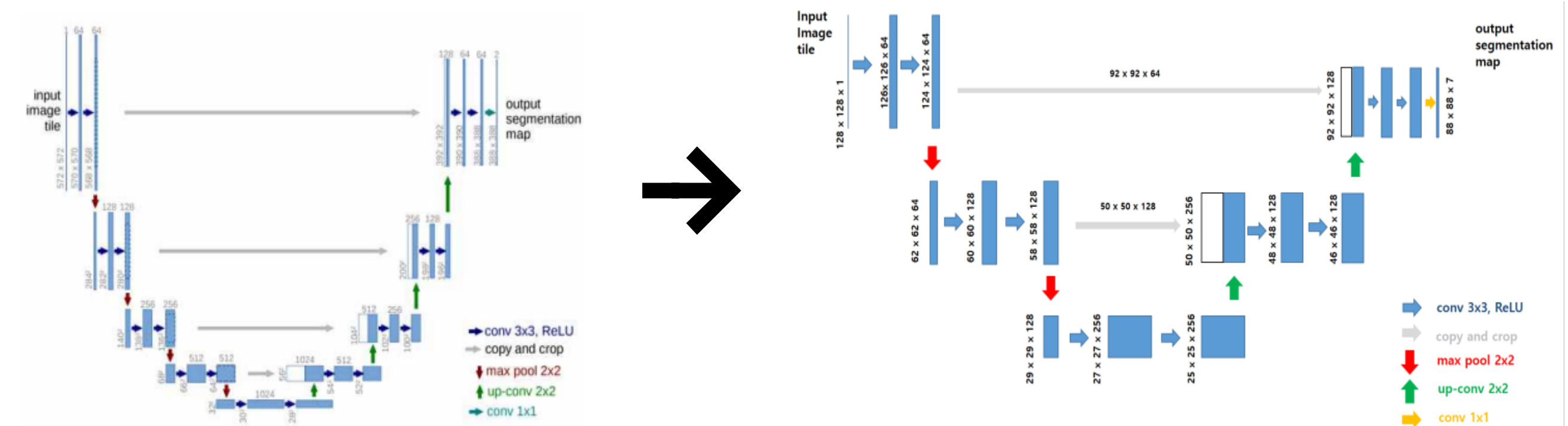
### 02 모델의 구조

U-NET 모델과 모델의 input image size(572 x 572)를 적용했을 때 실습환경인 kaggle notebook에서 할당된 memory 사용량 초과로 인한 **ResourceExhaustedError**가 발생하였습니다.

### 이미지 사이즈 변경

572 x 572 x 1 => 128 x 128 x 1로 변경.

### 기존의 U-NET 모델의 깊이를 변형



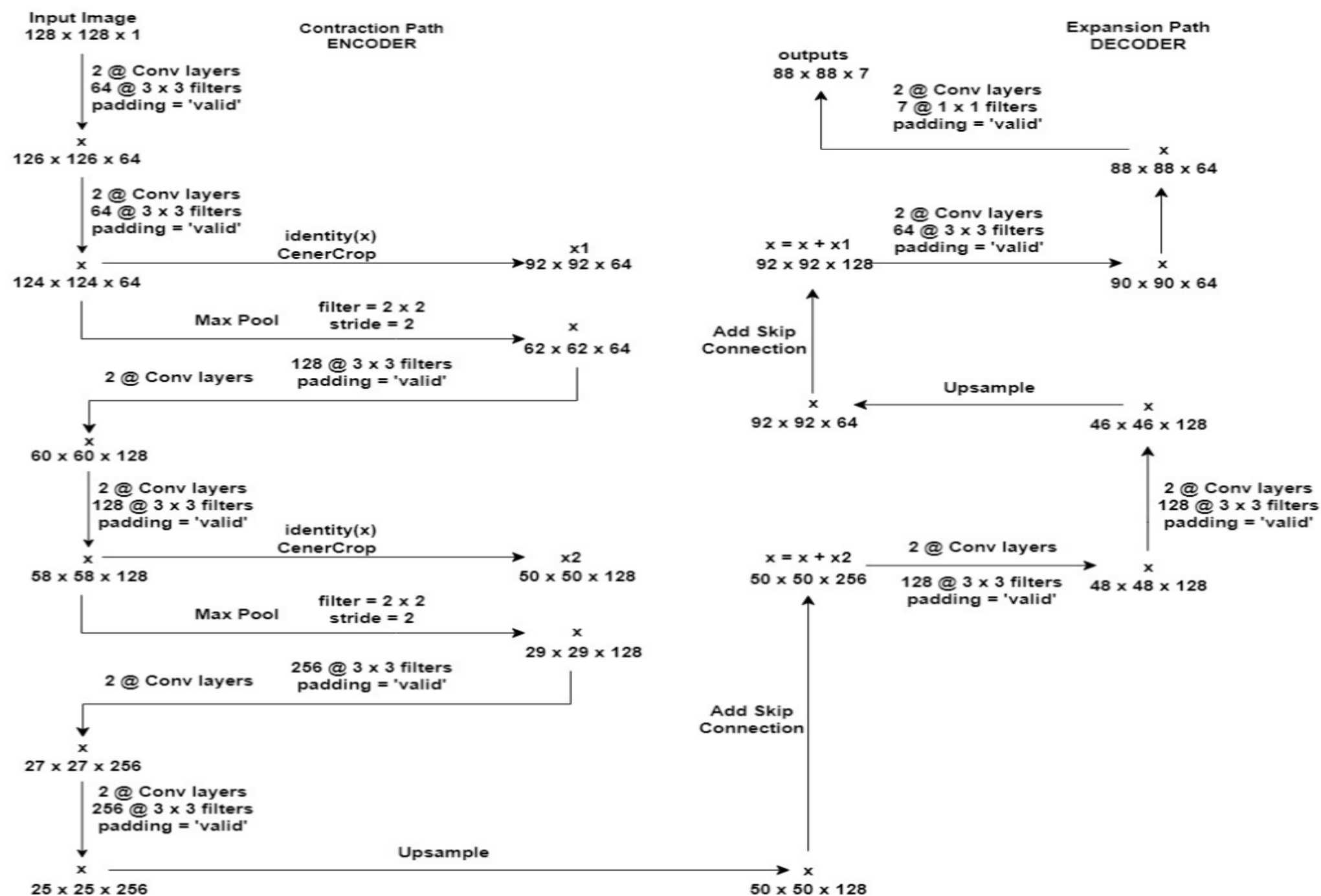
# 02 모델 구현

# Modeling

## 01 모델 설계

## 02 모델의 구조

## 모델 설계



# 02 모델 구현

## Modeling

### 01 모델 설계

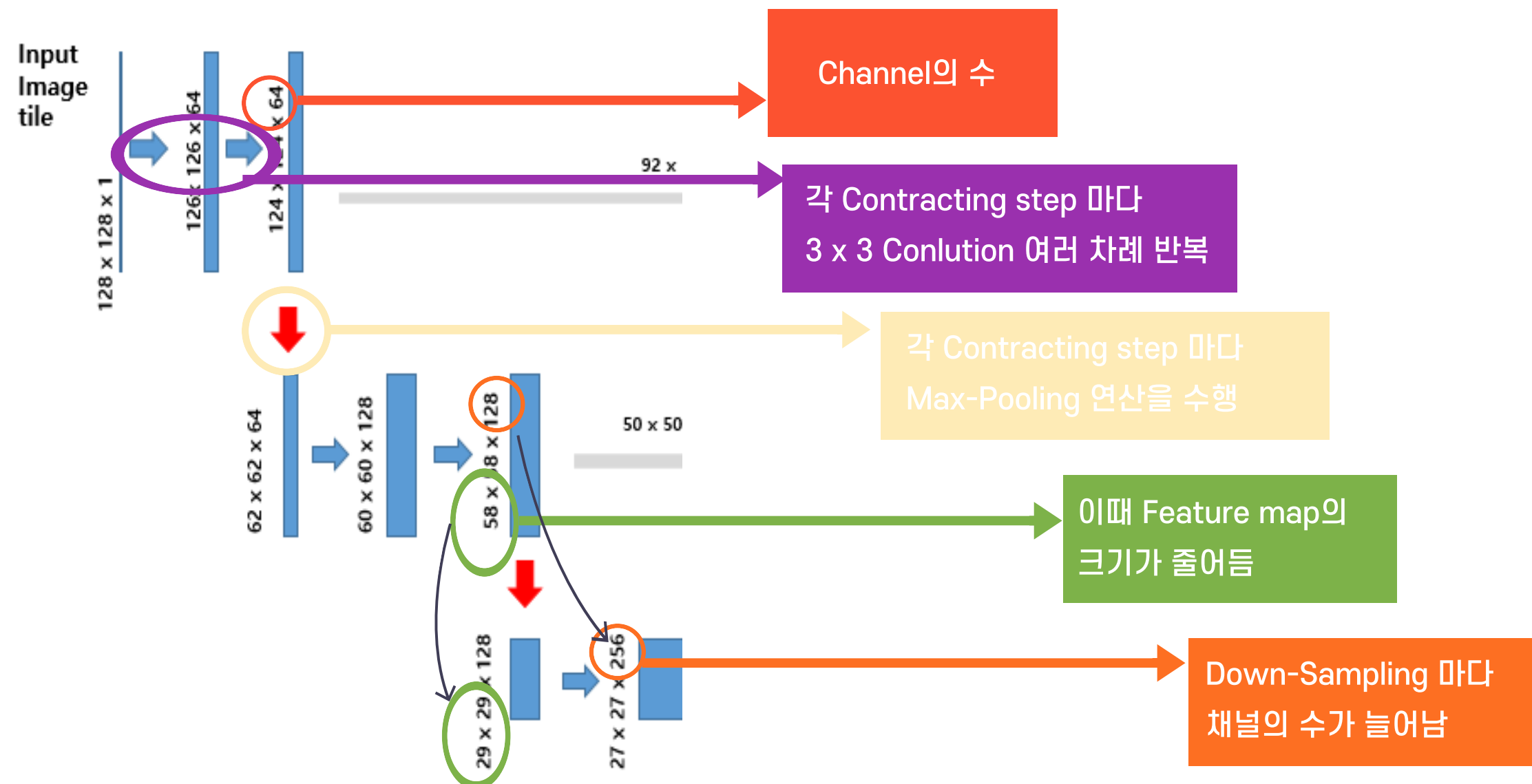
### 02 모델의 구조

#### 모델의 구조

1) Contracting Path : 이미지의 context 포착

-> 즉, 이미지에서 전체적인 구성을 보는 것입니다.

이미지에서 어떤 물체가 있는지에 대한 정보 등을 얻게 됩니다.



# 02 모델 구현

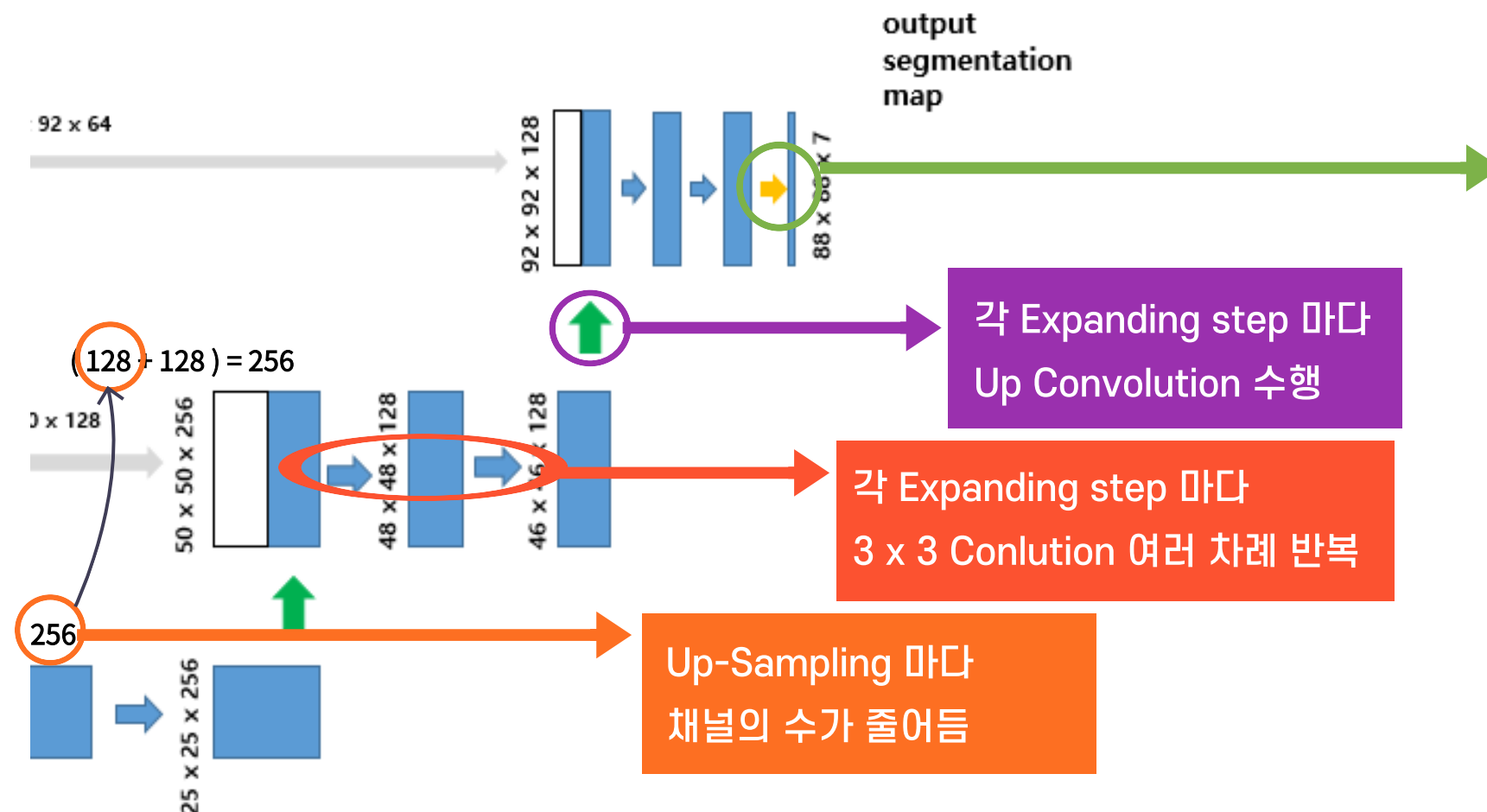
## Modeling

### 모델의 구조

#### 01 모델 설계

#### 02 모델의 구조

- 2) Expansive Path : Expansive Path에서의 역할은 Localization 입니다. 즉, 물체의 위치를 찾는 것입니다.  
얇은 레이어의 Feature Map을 결합.



마지막 Layer에 비선형성 예측을 위해  
1 x 1 Convolution 연산을 추가

1x1 Convolution Layer를 통과한 후 각  
픽셀이 어떤 Class에 해당하는지에 대한  
정보를 나타내는 3차원(Width x Height  
x Class) 벡터가 생성됨

# 02 모델 구현

## Modeling

### 모델의 구조

#### 01 모델 설계

#### 02 모델의 구조

copy and crop을 통해 contracting path에서 CenterCrop된 feature를 copy한 다음 expanding path의 대칭되는 계층에 concatenation을 했습니다 -> skip connection

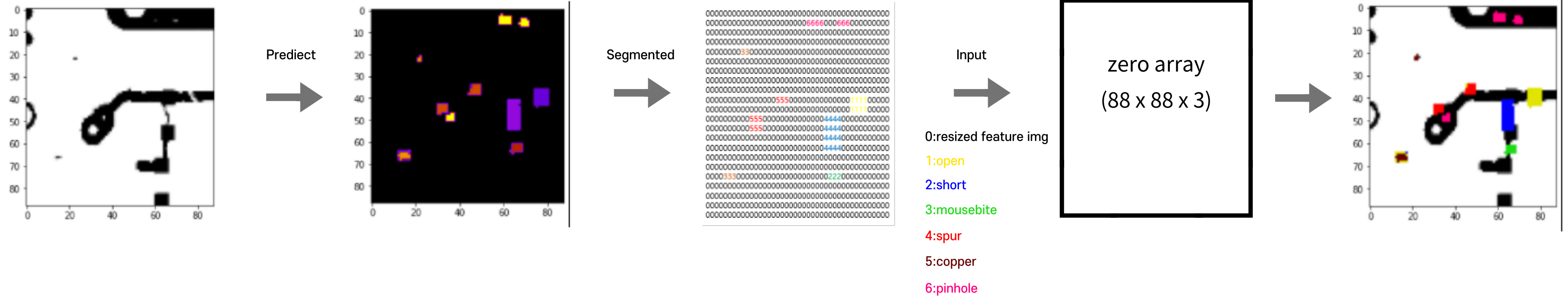
모델을 Encoder-Decoder 관점에서 살펴보면 인코더에서는 입력 이미지가 압축되다 보면 위치 정보가 어느 정도 손실되게 됩니다.

그렇게 되면 다시 원본 이미지 크기로 복원하는 과정 속에서 정보의 부족 때문에 원래 물체가 있었던 위치에서 어느 정도 이동이 발생하게 됩니다.

이런 복원 과정에 skip connection을 사용하게 되면 원본 이미지의 위치 정보를 추가적으로 전달받는 셈이 되므로 비교적 정확한 위치를 복원할 수 있게 되고 segmentation 결과도 좋아지게 됩니다.

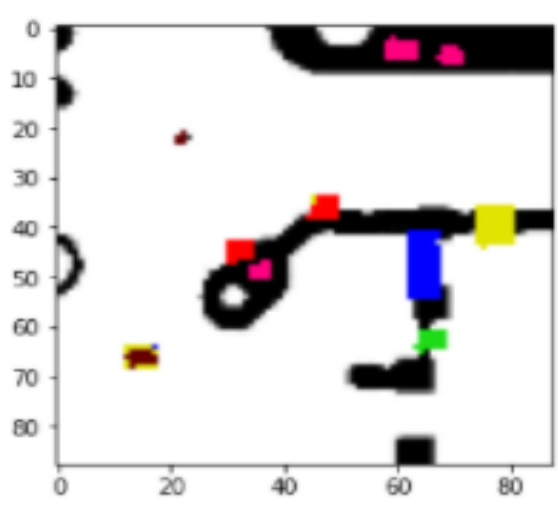
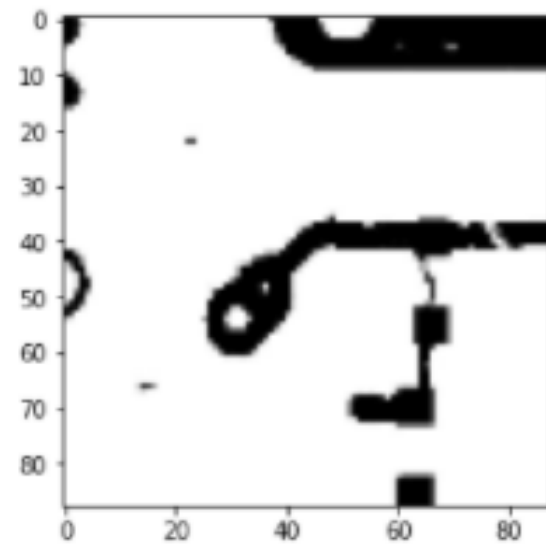
# 03 | 결과

# Result

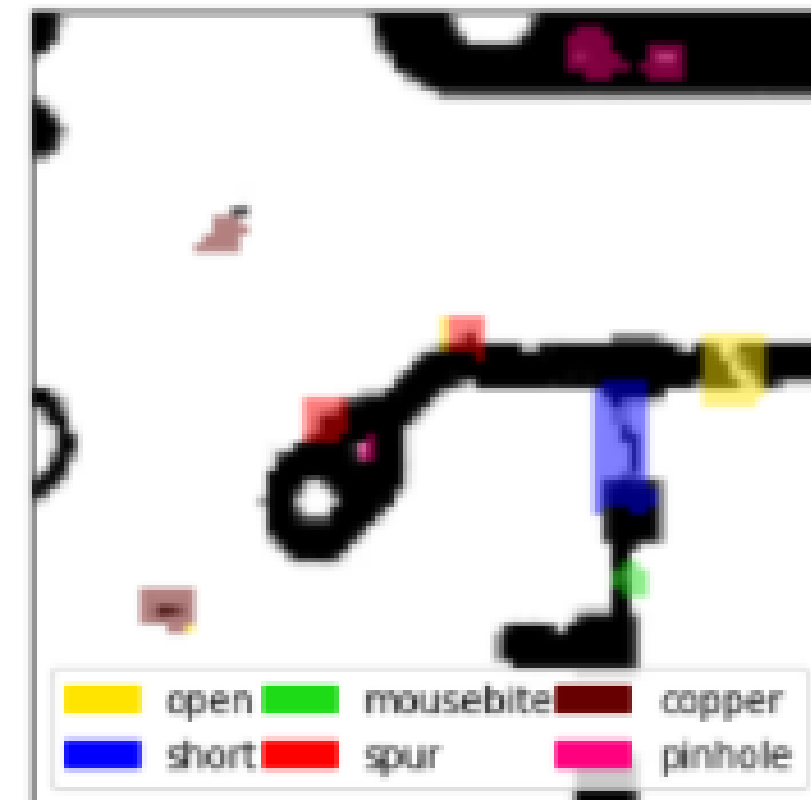


Semantic Segmentation을 통해 이미지 각 픽셀이 어느 클래스에 속하는지 예측

# Result Visualization



blend image





04 |

참조

# References

---

[1] <https://arxiv.org/abs/1505.04597>

[2] <https://joungeekim.github.io/2020/09/28/paper-review/>

[3] <https://m.blog.naver.com/j005580/221882044894>

[4] <https://gaussian37.github.io/vision-segmentation-unet/>

[5] <https://juseong-tech.tistory.com/2>

[6] <https://kuklife.tistory.com/118?category=872136>

---

감사합니다

---