


# 프로젝트 최종보고회

파손된 컨테이너 판별예측 프로젝트

# INDEX



I. 서론(INTRODUCTION)

II. 컨테이너 운송과 파손

III. 데이터 수집·증가·전처리

IV. 알고리즘 적용 결과

1. 머신러닝

2. 딥러닝

1) CNN

2) 전이학습

V. 결론(CONCLUSION)

# I .INTRODUCTION

1. 프로젝트 개요
2. 프로젝트 진행 일정

# 1. 프로젝트 개요

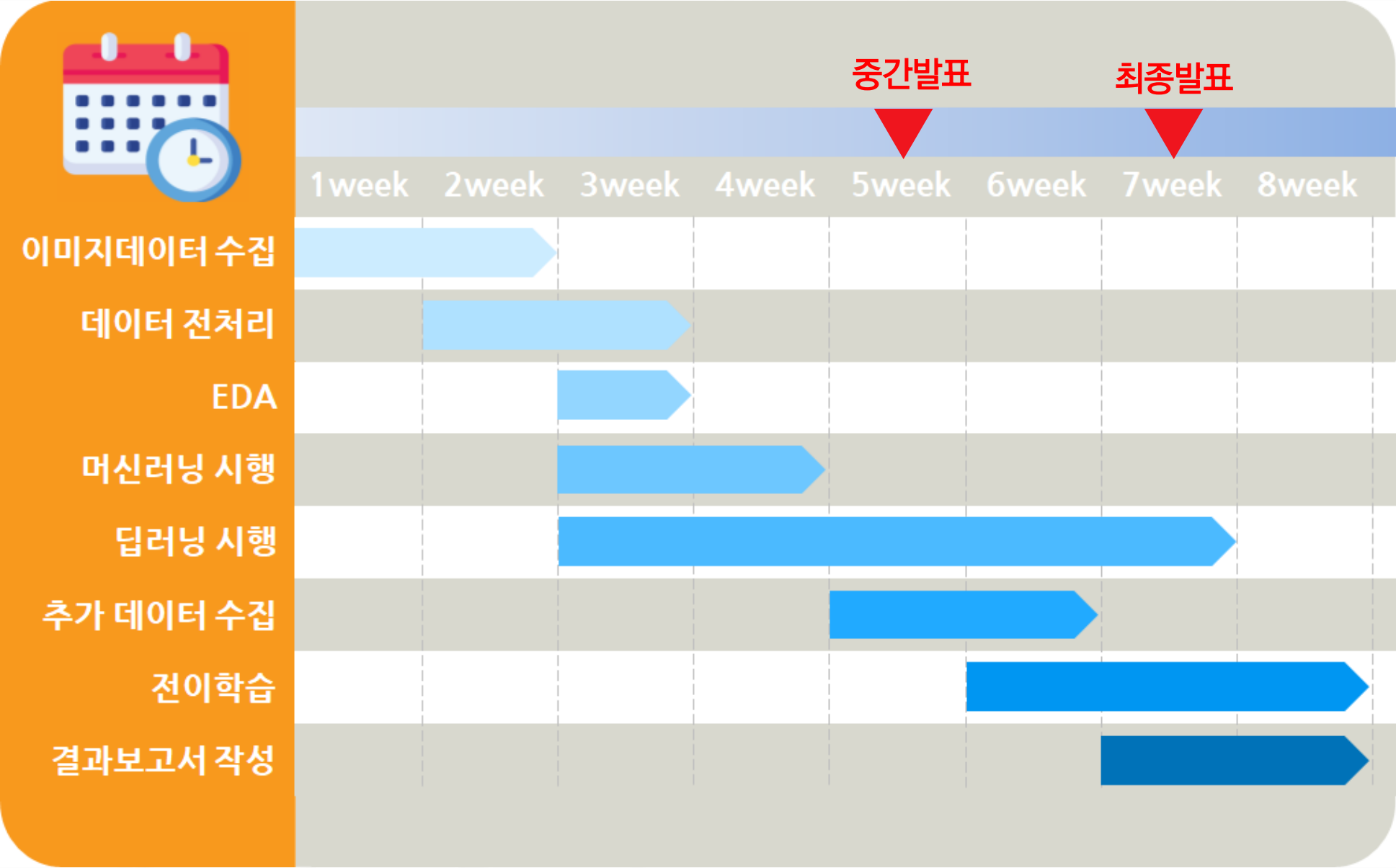
**프로젝트명** | 파손된 컨테이너 판별예측 프로젝트

**기간** | '21. 9. 6.(월) ~ '21.11. 5.(금), 8주간

**목적** | 파손 컨테이너 데이터 세트를 구축하고, 컨테이너 파손을 구분하는 이미지 분석 AI 모델을 개발

**기대 효과** | 컨테이너 파손 여부 자동 판별을 통한 효율적인 컨테이너 보수·수리 프로세스 수립 기여  
공급사슬 내 지점별 컨테이너 파손 여부 파악을 통한 물류 가시성 확보

## 2. 프로젝트 진행 일정



# II. 컨테이너 운송과 파손

1. 항만과 컨테이너
2. 컨테이너 파손 원인



# 1. 항만과 컨테이너



## 컨테이너 조작장(CFS)

- 컨테이너 적입, 적출이 이루어지는 공간
- 적재컨테이너 검사 진행

## 컨테이너 야드(CY)

- 컨테이너의 하역이 이루어지는 공간
- 공컨테이너 검사, 수리, 보수 작업 진행



## 2. 컨테이너 파손 원인



▲ 무리한 하역으로 인한 파손



▲ 지게발로 인한 파손



▲ 하중을 견디지 못해 파손



▲ 무리한 하역으로 인한 파손



▲ 불균형 적하로 인한 파손



▲ 운송 중 너울로 인한 전복



# III. 데이터 수집·증가·전처리

1. 데이터 수집(SCRAPING)

2. 배경 제거 작업(CROPPING)

3. 현장 촬영(ON-SITE EXPERIMENT)

4. 데이터 증가 작업(AUGMENTATION)

5. 데이터 전처리(DATA PRE-PROCESSING)

# 1. 데이터 수집(SCRAPING)

**수집 데이터** | 정상 및 파손 컨테이너 이미지

(주요 검색어: container, damaged container, cargo, container accident 등)

**방법** | 크롤링 코드 작성 후, 각 검색 플랫폼의 html 태그를 확인·수정하여 스크래핑을 수행  
보안 문제로 스크래핑이 불가능한 경우, 직접 다운로드하여 이미지 데이터 수집

**검색 플랫폼** | 한국 검색엔진: 네이버, 다음, 나무위키

외국 검색엔진: Google, Yahoo, Bing, Yandex, DuckDuckGo, Baidu ...

외국 이미지 전문 사이트: Unsplash, Pexels, Pixabay

**수집 결과** | 576장 수집(정상 299장, 파손 277장) (중복된 사진 제거)

## 2. 배경 제거 작업(CROPPING)

**목 적** | 인식하고자 하는 객체를 배경으로부터 분리하고, 컨테이너 객체를 명확히 하여 알고리즘에게 학습시키기 위하여 객체 주변의 배경을 제거하는 작업

**방 법** | PhotoShop, Illustrator, PowerPoint의 배경 제거 방법을 활용



### 3. 현장 촬영(ON-SITE EXPERIMENT)

**목 적** | 사진 데이터 추가적인 확보를 위한 CFS 내 장치된 컨테이너 사진 촬영

**방문 일시** | '21. 10. 21(수) 13:00 ~ 16:00

**방문 장소** | DW국제물류센터(운영팀 이정한 과장)  
부산크로스독(창고관리팀 신태용 부장)

**활동 결과** | 정상 컨테이너 사진 약 580여 장 촬영

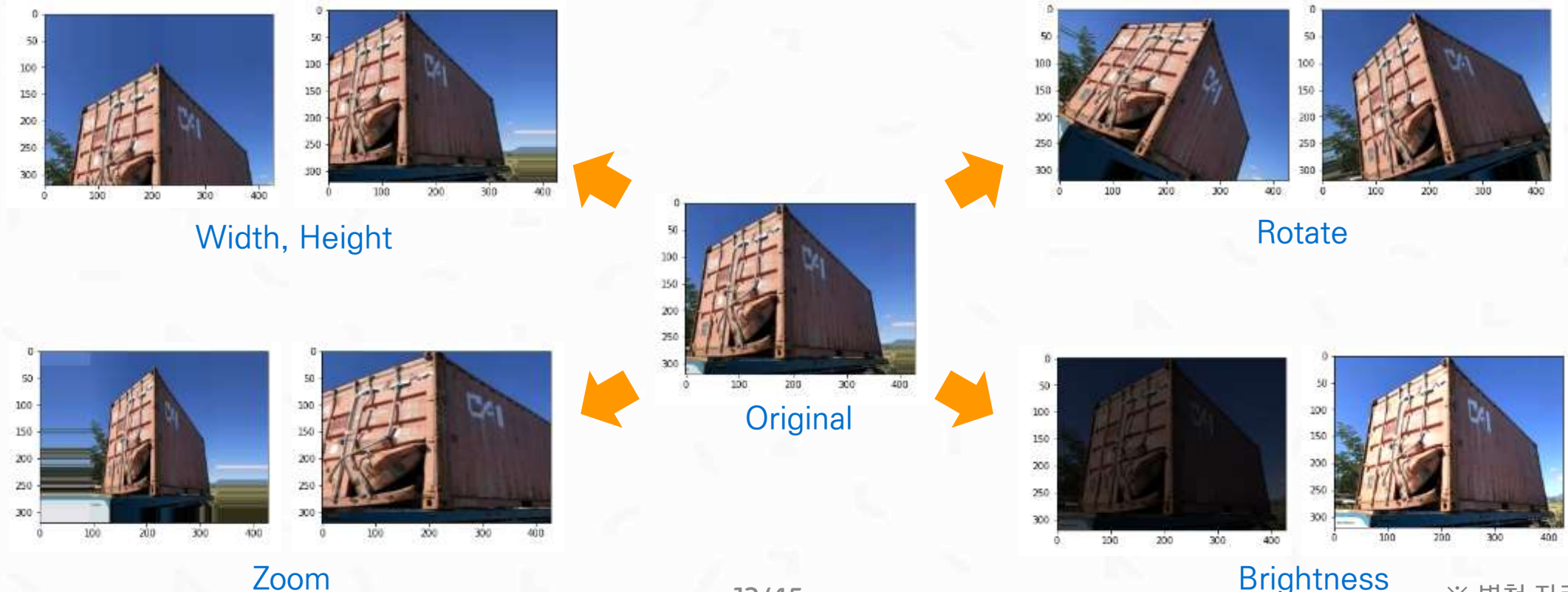




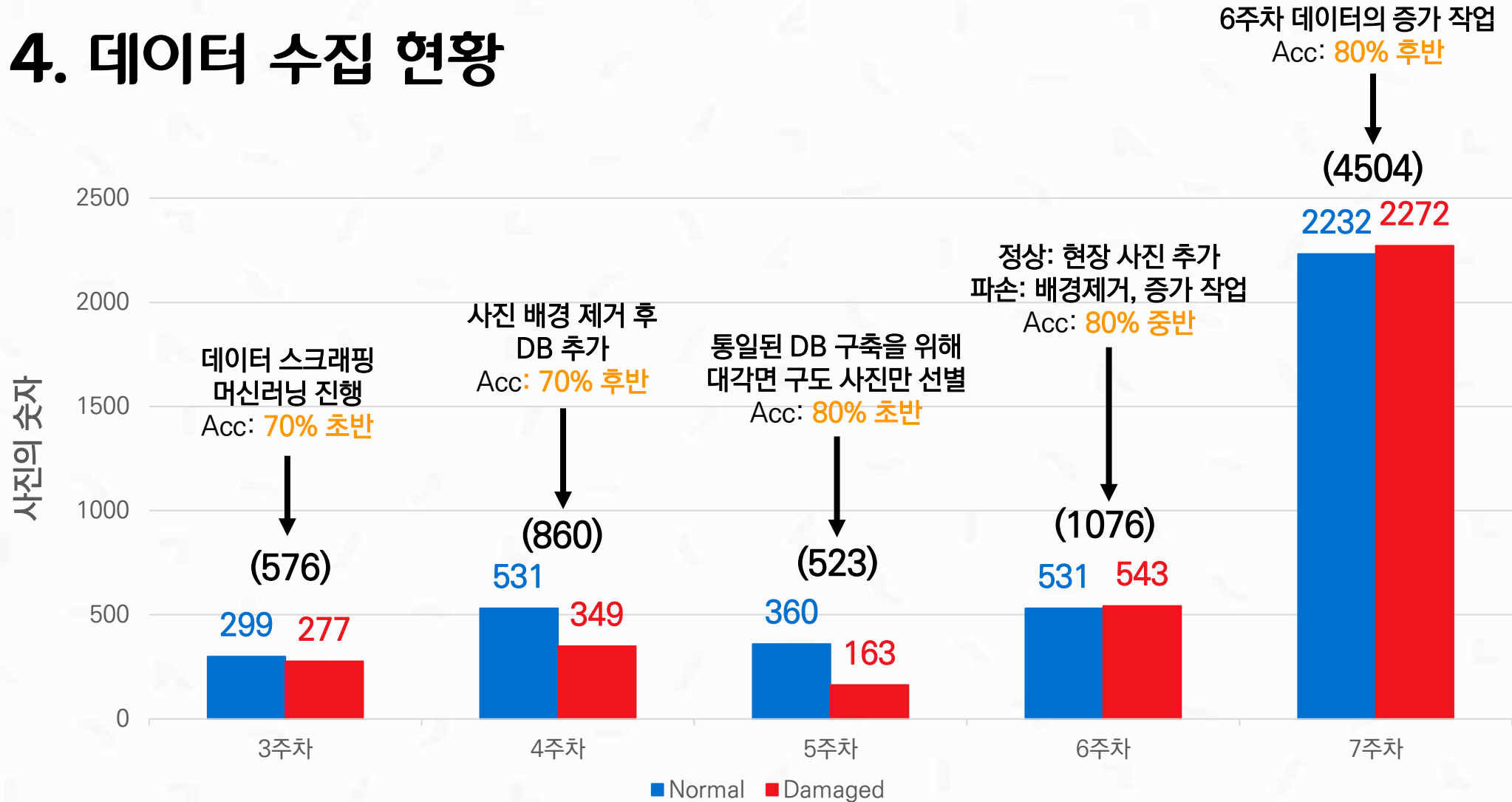
## 4. 데이터 증가 작업(AUGMENTATION)

**목적** | 사진 데이터의 양이 충분하지 못할 경우, 기존의 사진을 변형하여 양을 늘리는 작업  
사진의 양을 늘려 알고리즘에 학습이 충분히 될 수 있도록 할 때 사용

**방법** | Keras의 ImageDataGenerator 함수 사용



## 4. 데이터 수집 현황



※ 1주차는 스크래핑 코드 작성 및 작동여부 확인, 2,3주차부터 스크래핑을 수행하여 데이터를 확정

## 5. 데이터 전처리(DATA PRE-PROCESSING)

**목적** | 사진 데이터를 머신러닝 알고리즘 및 AI가 학습할 수 있게, 픽셀 값으로 변환

**순서** | ① 사진 데이터 열기 (Python Image Library의 Image 함수 활용)  
② 사진 Red, Green, Blue 채널 분리 후 픽셀 값 반환 (Numpy 라이브러리 활용)  
③ 픽셀값을 255로 나누어 표준화  
→ 4차원의 텐서 획득 (사진의 수, 128, 128, 3), 인풋 데이터로 저장  
④ 각 사진의 컨테이너 파손 여부를 알려주는 타겟 벡터 생성·저장

**데이터 세트** | 대각면 구도 사진 4504장(정상 컨테이너 2232장(50.4%), 손상 컨테이너 2272장(49.6%))  
훈련세트 3603장(79.9%), 테스트 세트 901장(20.1%) 구성(sklearn의 train\_test\_split 함수 사용)

# IV. 알고리즘 적용 결과

1. 머신러닝

2. 딥러닝



알고리즘 적용 결과

# 1. 머신러닝

K-최근접 이웃 분류

로지스틱 회귀

앙상블-랜덤포레스트

앙상블-XGBOOST

※ 본 결과는 멤버들의 머신러닝 학습을 위해 시행한 것으로, 적용 결과는 좋지 않음을 미리 밝힙니다.

# 1) 머신러닝

알고리즘	Parameters	Train score	Test score	적합 여부
K-최근접 이웃 분류	n=100	0.7124	0.7069	과소적합
로지스틱 회귀	C=1E-8	0.5193	0.5172	과소적합
랜덤 포레스트	default	1.0000	0.7566	과대적합
XGBoost	n_estimators=200, max_depth=3, eval_metric="mlogloss"	1.0000	0.7490	과대적합

→ 머신러닝 방법이 이미지 데이터에 적용하기 어렵다고 판단  
따라서, 이미지 데이터 분류에 적합한 딥러닝 알고리즘을 수행

알고리즘 적용 결과

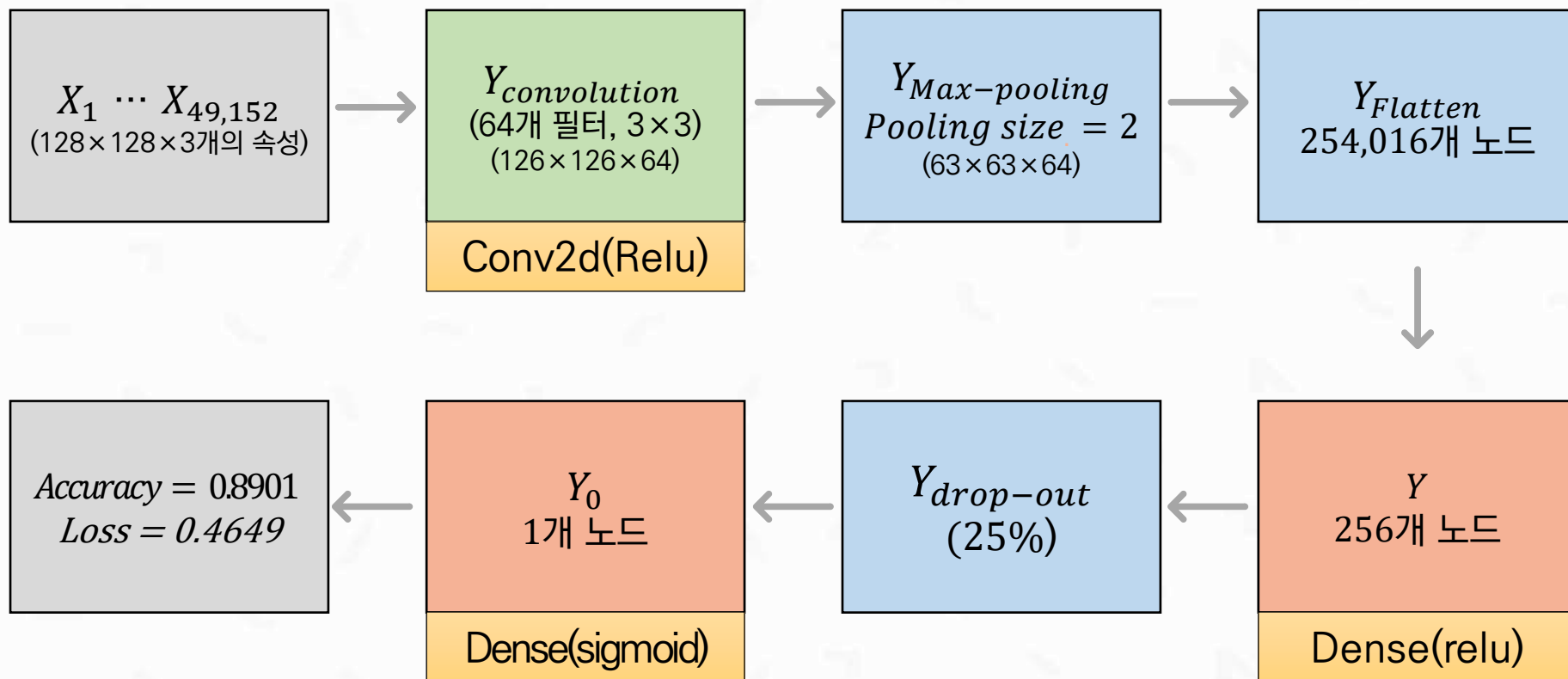
## 2. 딥러닝

- 1) CNN
- 2) 전이학습

# 1) CNN(CONVOLUTION NEURAL NETWORK)

**정의** | 컨볼루션 층과 풀링 층을 통하여 데이터의 특징을 컴퓨터가 스스로 학습하는 알고리즘  
현존하는 알고리즘 중 이미지나 글씨에 강력한 성능을 지님

[ 프로젝트 기본 모델 구조 설계 ]



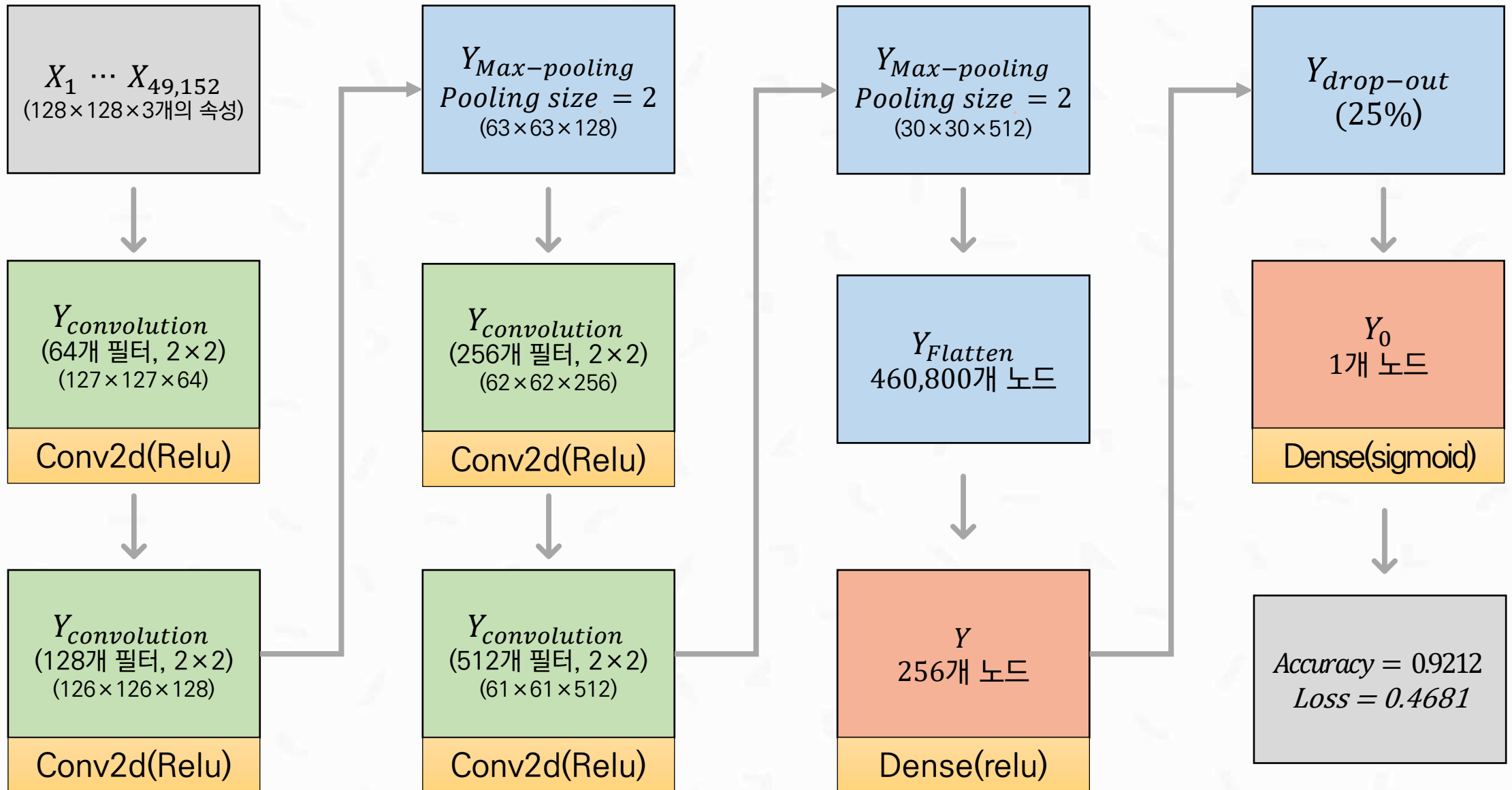


# CNN 수행 결과

모델	conv2d			Dense		맥스 풀링	드롭 아웃	출력층	Accuracy	Loss
	필터 수	필터 사이즈	활성화 함수	노드 수	활성화 함수					
Model 1	32	3 × 3	Relu	256	Relu	2	0.25	sigmoid	0.8812	0.4505
Model 2	64	3 × 3	Relu	256	Relu	2	0.25	sigmoid	0.8901	0.4649
Model 3	64	4 × 4	Relu	256	Relu	2	0.25	sigmoid	0.8701	0.5508
Model 4	64	2 × 2	Relu	256	Relu	2	0.25	sigmoid	0.8957	0.4540
Model 5	64 128 256 512	2 × 2	Relu	256	Relu	2	0.25	sigmoid	0.9212	0.4681

※ 학습 횟수(Epochs) : 30, 1회당 샘플 처리 개수(Batch\_size) : 64

# CNN 최적 모델 구조(MODEL 5)



# CNN MODEL 5 훈련 결과

## Model.fit() & Model.evaluate()

```
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=30, batch_size=67)
```

```
Epoch 21/30
54/54 [=====] - 14s 261ms/step - loss: 0.0847 - accuracy: 0.9756 - val_loss: 0.6834 - val_accuracy: 0.8713
Epoch 22/30
54/54 [=====] - 14s 260ms/step - loss: 0.0546 - accuracy: 0.9833 - val_loss: 0.4364 - val_accuracy: 0.9079
Epoch 23/30
54/54 [=====] - 14s 262ms/step - loss: 0.0191 - accuracy: 0.9939 - val_loss: 0.4554 - val_accuracy: 0.9123
Epoch 24/30
54/54 [=====] - 14s 263ms/step - loss: 0.0076 - accuracy: 0.9986 - val_loss: 0.4643 - val_accuracy: 0.9212
Epoch 25/30
54/54 [=====] - 14s 261ms/step - loss: 0.0055 - accuracy: 0.9992 - val_loss: 0.5459 - val_accuracy: 0.9212
Epoch 26/30
54/54 [=====] - 14s 268ms/step - loss: 0.0055 - accuracy: 0.9981 - val_loss: 0.5947 - val_accuracy: 0.9134
Epoch 27/30
54/54 [=====] - 15s 273ms/step - loss: 0.0130 - accuracy: 0.9969 - val_loss: 0.4869 - val_accuracy: 0.9145
Epoch 28/30
54/54 [=====] - 14s 268ms/step - loss: 0.0054 - accuracy: 0.9983 - val_loss: 0.5955 - val_accuracy: 0.9156
Epoch 29/30
54/54 [=====] - 14s 263ms/step - loss: 0.0229 - accuracy: 0.9922 - val_loss: 0.4769 - val_accuracy: 0.9057
Epoch 30/30
54/54 [=====] - 14s 261ms/step - loss: 0.0237 - accuracy: 0.9914 - val_loss: 0.4681 - val_accuracy: 0.9212
```

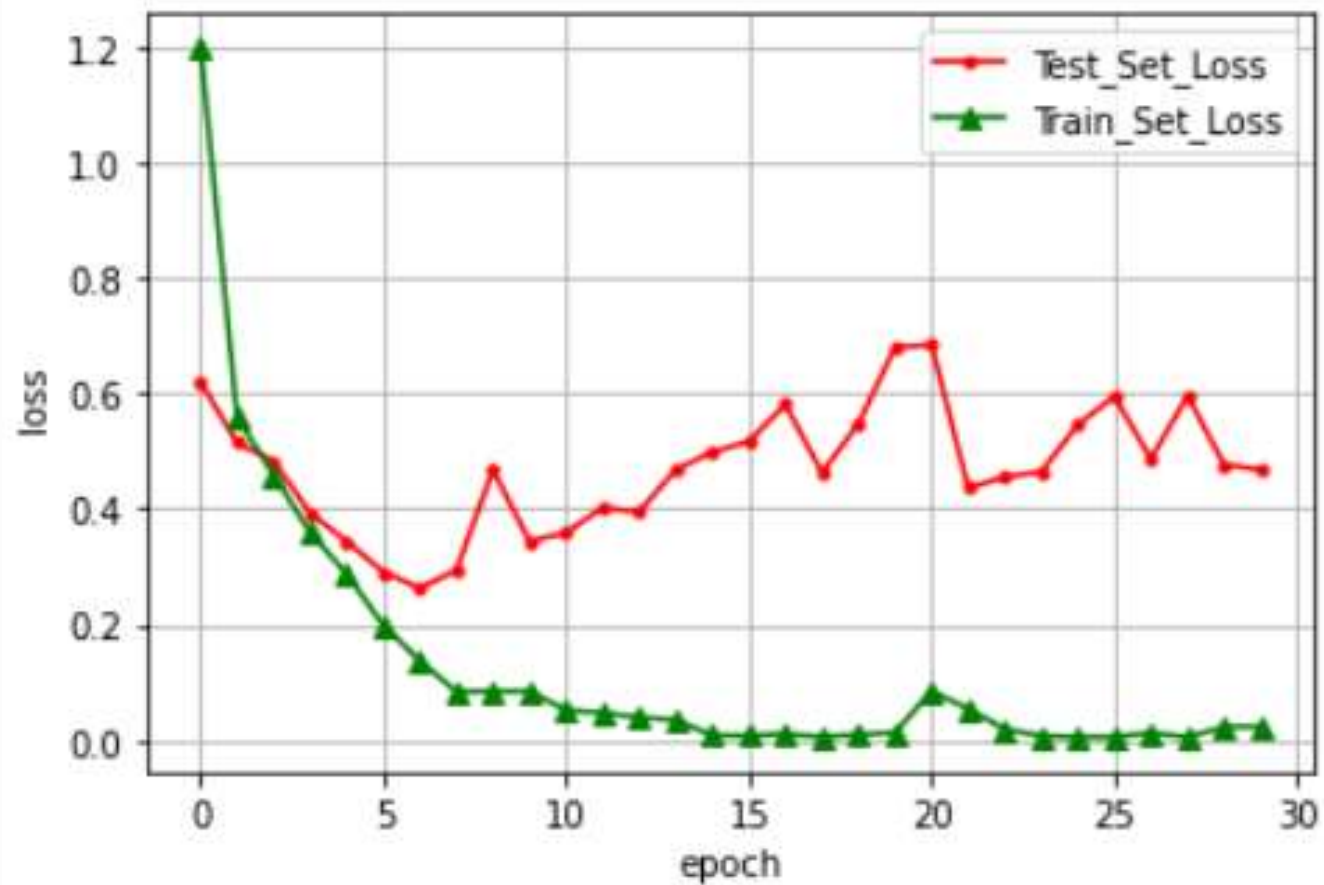
```
# 테스트 세트에 적용해보기
```

```
print('Wn Test Accuracy: %.4f' % (model.evaluate(X_test, Y_test)[1]))
```

```
29/29 [=====] - 2s 39ms/step - loss: 0.4681 - accuracy: 0.9212
```

# CNN MODEL 5 훈련 결과

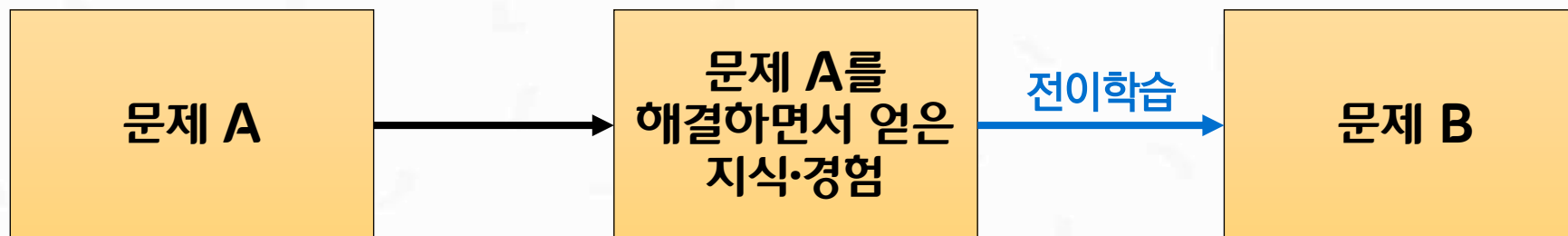
[ 프로젝트 Model 5 훈련 과정 그래프(Loss 기준) ]





## 2) 전이학습(TRANSFER LEARNING)

**정의** | 큰 데이터 셋으로 훈련된 모델의 가중치를 가져와, 사용자가 구축한 데이터에 적용하여 학습하는 방식  
이미 훈련된 Backbone을 활용하여 특성을 뽑아내고, 본인의 데이터에 맞게 완전연결층을 조정하는 방식



자료: 서지영(2021), 딥러닝 텐서플로 교과서, p.165.

# ① ALEXNET

**모델 개요** | ILSVC(이미지넷 이미지 인식 대회)에서 2012년에 우승한 알고리즘

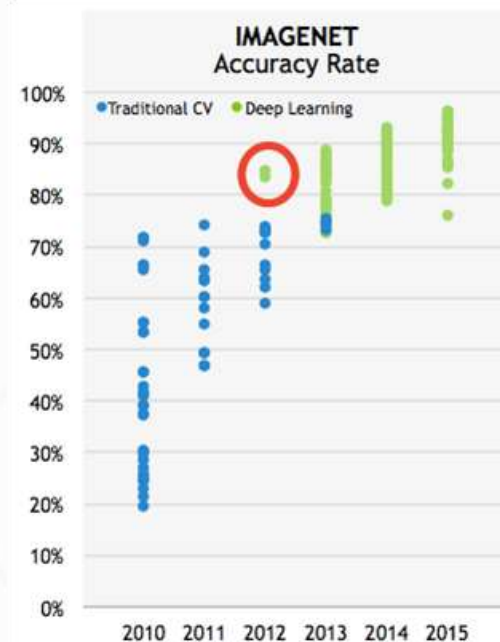
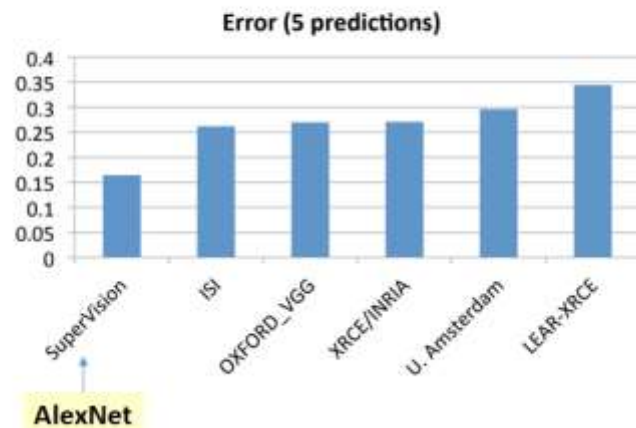
**모델 특징** | ① 컴퓨터 비전(Computer Vision)에서 딥러닝 기법을 최초 사용

② GPU 2대를 사용하여 연산속도 가속화

③ 활성화함수 'ReLU' 사용

**모델 구성** | 합성곱 층 5개, 완전연결층 3개

Ranking of the best results from each team



# ALEXNET 훈련

```
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath = './model/alexnet.hdf5'

checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)
```

# 모델의 실행

```
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test),
                    epochs=50, batch_size=128, verbose=1,
                    callbacks=[early_stopping_callback, checkpointer])
```

Epoch 1/50

29/29 [=====] - 7s 141ms/step - loss: 55.8199 - accuracy: 0.5132 - val\_loss: 0.6929 - val\_accuracy: 0.5105 - accuracy

Epoch 00001: val\_loss improved from inf to 0.69292, saving model to ./model/alexnet.hdf5

Epoch 2/50

29/29 [=====] - 3s 100ms/step - loss: 0.6934 - accuracy: 0.4886 - val\_loss: 0.6904 - val\_accuracy: 0.5050

.

.

.

Epoch 00036: val\_loss did not improve from 0.24861

Epoch 37/50

29/29 [=====] - 3s 99ms/step - loss: 0.1296 - accuracy: 0.9606 - val\_loss: 0.3351 - val\_accuracy: 0.9267

Epoch 00037: val\_loss did not improve from 0.24861

# ALEXNET 훈련결과

```
# 테스트 세트에 적용해보기
```

```
print('\n Test Accuracy: %.4f' % (model.evaluate(X_test, Y_test)[1]))
```

```
29/29 [=====] - 1s 12ms/step - loss: 0.3351 - accuracy: 0.9267
```

```
Test Accuracy: 0.9267
```

## Checkpoint된 최선의 결과 출력

```
from tensorflow.keras.models import load_model  
model2 = load_model('./model/alexnet.hdf5')
```

```
# 테스트 세트에 적용해보기
```

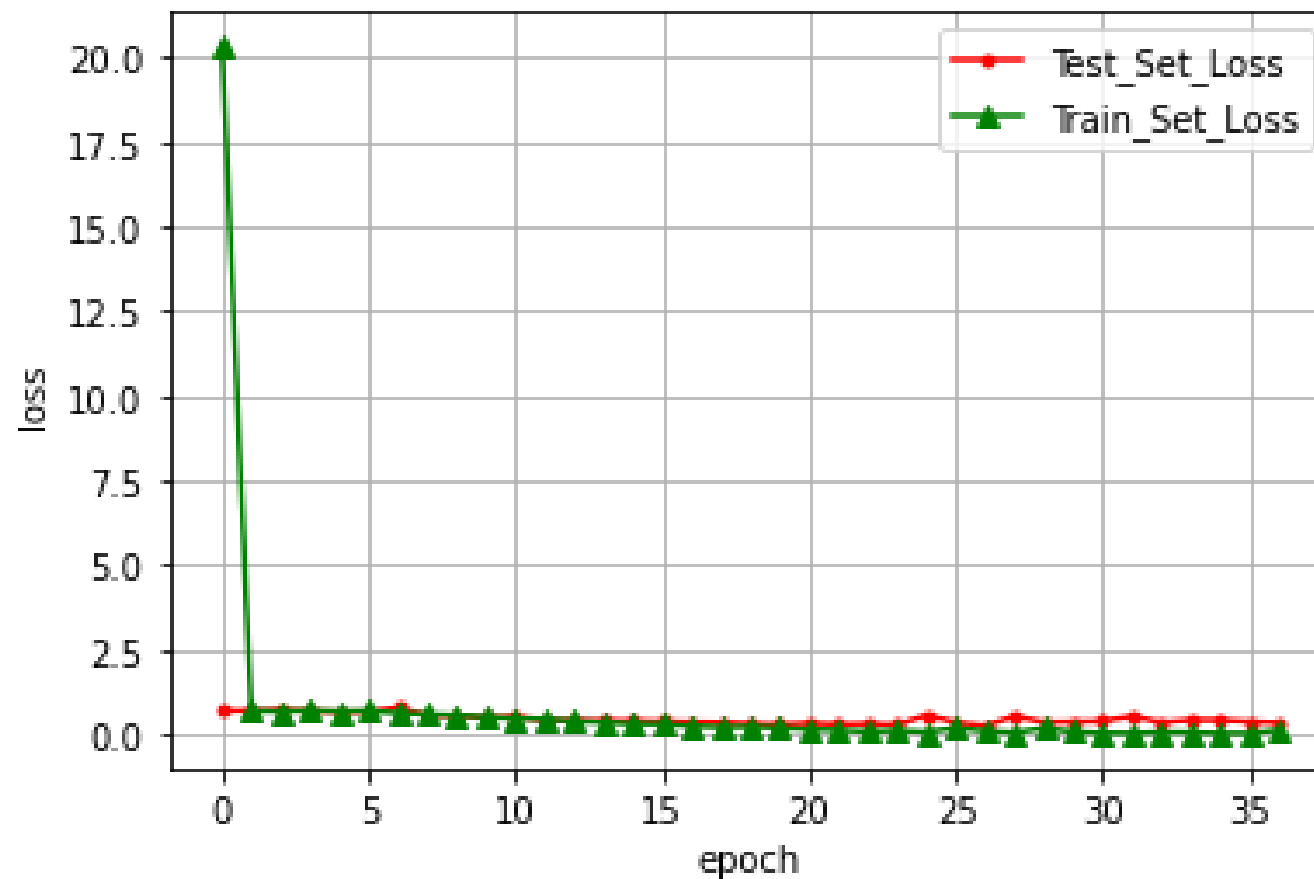
```
print('\n Test Accuracy: %.4f' % (model2.evaluate(X_test, Y_test)[1]))
```

```
29/29 [=====] - 0s 12ms/step - loss: 0.2486 - accuracy: 0.9334
```

```
Test Accuracy: 0.9334
```

# ALEXNET 훈련결과

[ AlexNet 훈련 과정 그래프(Loss 기준) ]



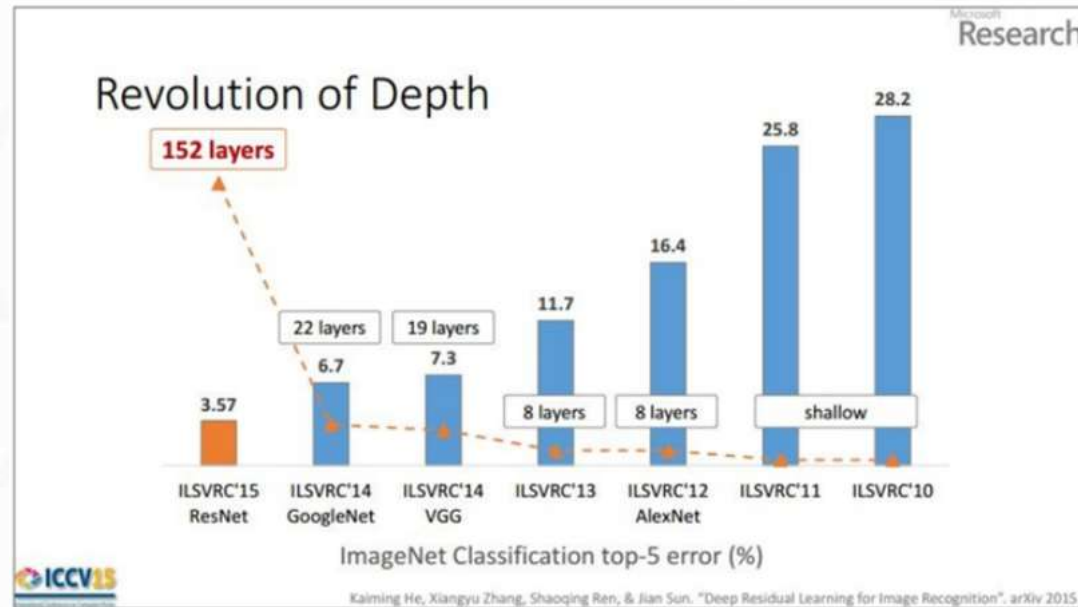
## ② VGG16(VISUAL GEOMETRY GROUP)

**모델 개요** | ILSVC(이미지넷 이미지 인식 대회)에서 2014년에 준우승한 알고리즘

**모델 특징** | ① CNN의 기본 아키텍처(합성곱층, 풀링층, 완전연결층)만 활용하여 모델을 구성

② 필터 사이즈가  $3 \times 3$ 으로 동일, 맥스풀링층도  $2 \times 2$ 로 동일

**모델 구성** | 합성곱 층 13개, 완전연결층 3개





# VGG-16 훈련

```
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath = './model/vgg16.hdf5'

checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# 모델의 실행
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_test),
                    epochs=50, batch_size=128, verbose=1,
                    callbacks=[early_stopping_callback, checkpointer])
```

```
Epoch 1/50
29/29 [=====] - 27s 477ms/step - loss: 1.4600 - acc: 0.5630 - val_loss: 0.6030 - val_acc: 0.7159
```

```
Epoch 00001: val_loss improved from inf to 0.60298, saving model to ./model/vgg16.hdf5
```

```
Epoch 2/50
29/29 [=====] - 10s 335ms/step - loss: 0.5541 - acc: 0.7419 - val_loss: 0.2574 - val_acc: 0.9123
```

·  
·  
·

```
Epoch 00032: val_loss did not improve from 0.02567
```

```
Epoch 33/50
```

```
29/29 [=====] - 10s 338ms/step - loss: 0.0167 - acc: 0.9938 - val_loss: 0.0529 - val_acc: 0.9889
```

```
Epoch 00033: val_loss did not improve from 0.02567
```

# VGG-16 훈련결과

## 마지막 epoch 기준 결과 출력

```
# 테스트 세트에 적용해보기
```

```
print('\n Test Accuracy: %.4f' % (model.evaluate(X_test, Y_test)[1]))
```

```
29/29 [=====] - 6s 68ms/step - loss: 0.0529 - acc: 0.9889
```

```
Test Accuracy: 0.9889
```

## Checkpoint된 최선의 결과 출력

```
from tensorflow.keras.models import load_model
```

```
model2 = load_model('./model/vgg16.hdf5')
```

```
# 테스트 세트에 적용해보기
```

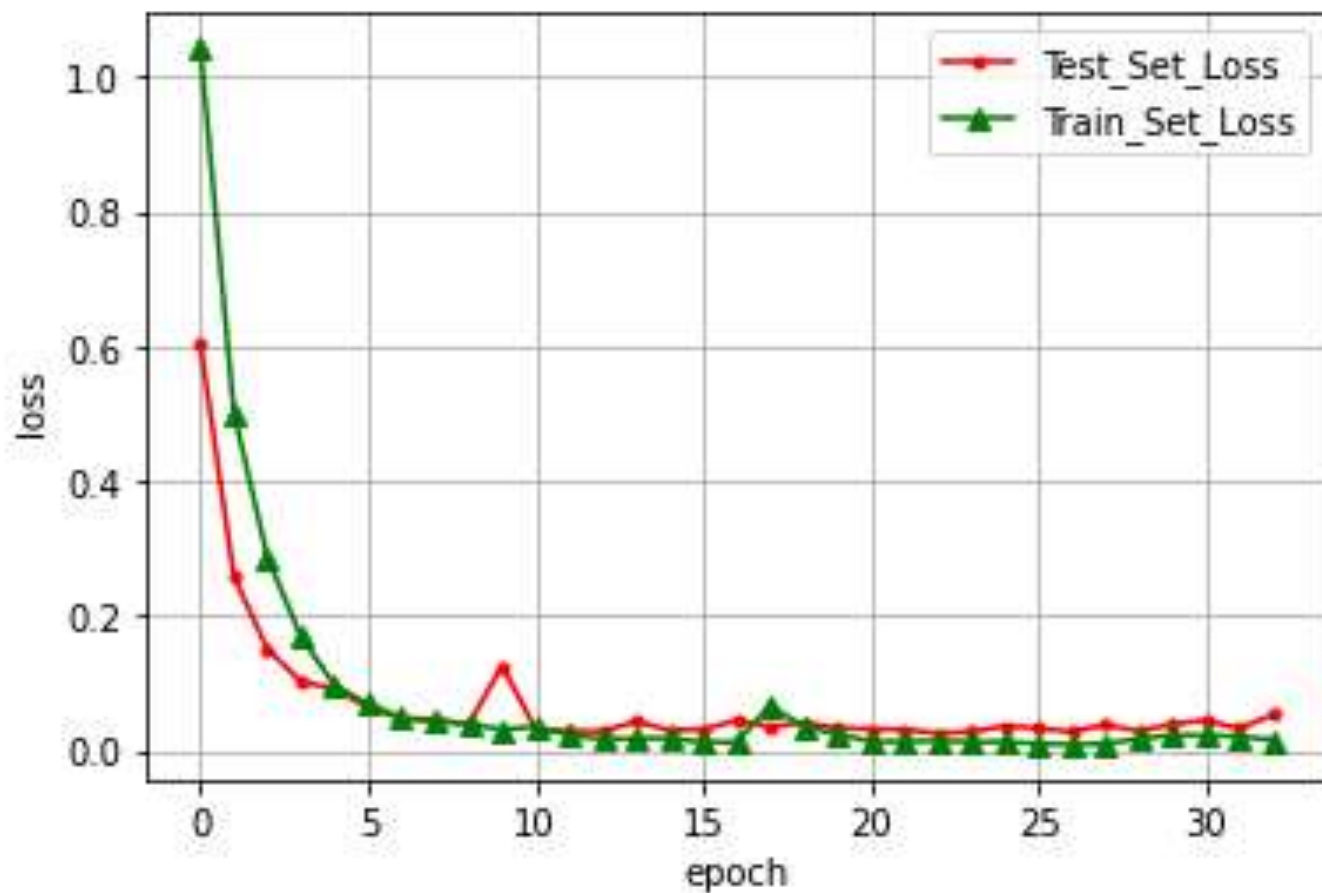
```
print('\n Test Accuracy: %.4f' % (model2.evaluate(X_test, Y_test)[1]))
```

```
29/29 [=====] - 2s 68ms/step - loss: 0.0257 - acc: 0.9933
```

```
Test Accuracy: 0.9933
```

# VGG-16 훈련결과

[ VGG-16 훈련 과정 그래프(Loss 기준) ]



# CNN·전이학습 수행 결과 종합

알고리즘	수행 방법	적용 모델	정확도	손실
CNN	직접 모델 설계	Final Model	0.9212	0.4681
		Best Model(4th)	0.8824	0.2974
AlexNet	아키텍처 직접 적용	Final Model	0.9267	0.3351
		Best Model(27th)	0.9334	0.2468
VGG16	이미지넷 가중치 적용	Final Model	0.9889	0.0529
		Best Model(23rd)	0.9933	0.0257

# V. CONCLUSION

1. 프로젝트 결과 및 활용 방안
2. LESSON LEARNED
3. DISCUSSION AND NEXT STEP

# 1. 프로젝트 결과 및 활용 방안

## 결과

1. 정상 컨테이너와 파손 컨테이너 이미지를 스크래핑을 통하여 데이터 세트 구축
2. 데이터 세트 학습을 통해 컨테이너 파손을 판별하는 이미지 분석 AI 모델 개발 (CNN 정확도 92.12%)
3. 전이학습을 활용, 컨테이너 파손을 구분하는 이미지 분석 AI 모델 개발 (AlexNet 정확도 93.34%, VGG16 정확도 99.33%)

## 활용 방안

1. 컨테이너 파손 여부를 자동으로 판별하여 효율적인 컨테이너 보수·수리 프로세스 수립에 기여
2. 공급사슬 내 지점별 컨테이너 파손 여부 파악을 통한 물류 가시성 확보



## 2. LESSON LEARNED

### 좋았던 점

1. 데이터 수집에서부터 결과 분석까지 프로젝트 전체 과정을 실습
2. 현장에서 실무를 담당하는 멘토와의 소통으로 프로젝트 진행 과정 경험하고 질의 응답 시간을 가졌던 것
3. 수업 내용 확장 뿐만 아니라, 수업에서 접하지 못한 이론들도 접하는 기회

### 교훈

1. 다양한 방법으로 상이한 결과를 얻는 과정을 통해 데이터의 특성마다 적합한 방법이 있다는 것을 깨달음
2. 모델 설계만큼 데이터 수집과 전처리 작업에 많은 시간이 필요하다는 것을 배움
3. 이미 널리 알려진 유명한 모델을 통해 결과를 도출해 보면서, 접근 방향성을 좀 더 넓힐 수 있다는 것을 알

### 3. DISCUSSION AND NEXT STEP

#### DISCUSSION

- 대각면 뿐만 아니라, 다른 각도 사진으로 높은 정확도를 가진 알고리즘을 만들 수 있을까?
- 파손의 유형에 따른 분류 알고리즘 체계도 만들 수 있을까?
- 책에서는 층을 작게 만들면서 정확도를 높였지만, 우리는 층을 크게 만들면서 정확도를 높였다. 층과 정확도의 관계가 무엇일까?

#### NEXT STEP

- 정면과 측면 사진을 섞었을 때도 높은 정확도를 가진 알고리즘을 구축하도록 데이터 증가를 해보겠다.
- 컨테이너 파손 유형은 상당히 다양하므로, 유형을 조사해서 클래스를 설정하고 알고리즘을 구축
- 컨테이너 검사관에 따르면, 파손 여부를 검사할 때 가장 중요하게 보는 것은 컨테이너 고리임.  
따라서 컨테이너 고리 사진을 부분적으로 추출하여 알고리즘을 구축하면 산업 현장에 많은 도움이 될 것이라 생각됨

**THANK YOU!**

Q&A