III장. AI훈민정음_2
Binary_Anomaly_Detection
(악성코드 이상탐지)

목 차

I. 개요

1. 소개 : 악성코드 이상탐지

Kaggle에서 제공하는 비악성코드 데이터를 오토인코더 기반의 U-Net 모델로 학습시켜 악성코드를 탐지하는 예제

1. 데이터셋

Malware as image Dataset :
악성코드 또는 비악성코드의 binary데이터를 이미지로 표현 시킨 데이터셋을 사용

이 중 282장의 이미지(500,500)들을 사용

2. 전처리 및 학습

전처리 :
데이터 이미지를 같은 크기로 변환,
이미지 픽셀값을 0~1 사이로 정규화

학습 :
오토인코더 기반의 U-Net 모델을 사용하여 학습

3. 추론 결과

추론 데이터가 비악성코드이면 benign을 출력, 악성코드이면 malware를 출력하여 이상탐지

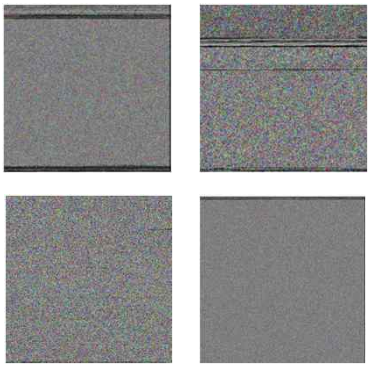
4. 기대 효과

악성코드를 탐지하여 악성코드 감염예방에 활용
신종 또는 변종 악성코드 탐지에 활용
시스템 로그를 탐지하여 시스템 오류 탐지에 활용

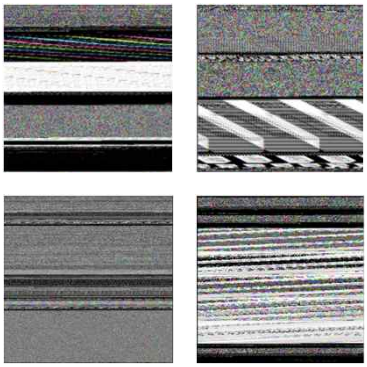
데이터셋

- 비악성코드(122장)와 악성코드(160장)의 binary데이터를 이미지로 표현한 데이터 셋
- 학습 : 비악성코드 이미지 100장
- 검증 : 비악성코드 이미지 22장, 악성코드 이미지 22장 사용

정상 데이터



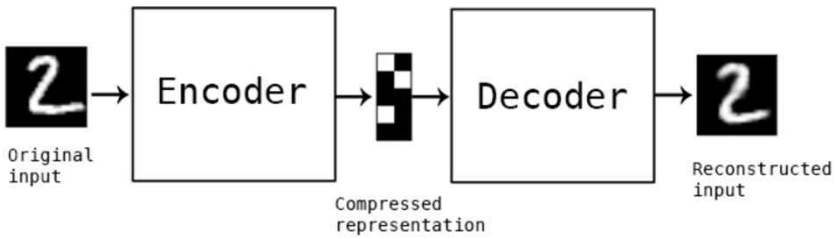
악성 데이터



I. 개요

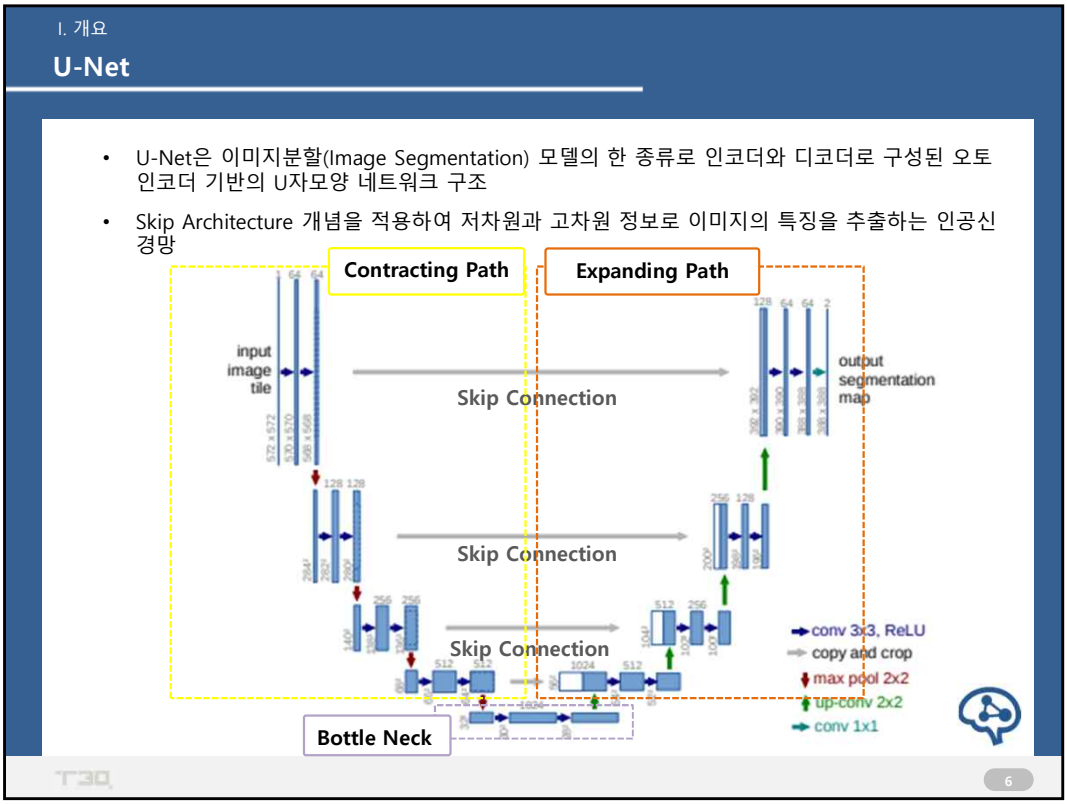
오토인코더

- 오토인코더(AutoEncoder)는 인코더와 디코더로 구성되어 있는 비지도 학습의 한 종류이며, 입력데이터가 학습에 정답데이터로 사용됨
- 즉, 입력 값이 주어지면 데이터를 압축하여 저차원의 잠재 벡터로 인코딩한 다음 잠재 벡터를 디코딩하여 입력 값과 출력 값을 동일하게 복원하도록 학습된 신경망



5

이미지 출처 : https://blog.keras.io/building-autoencoders-in-keras.html(케라스 공식홈페이지)

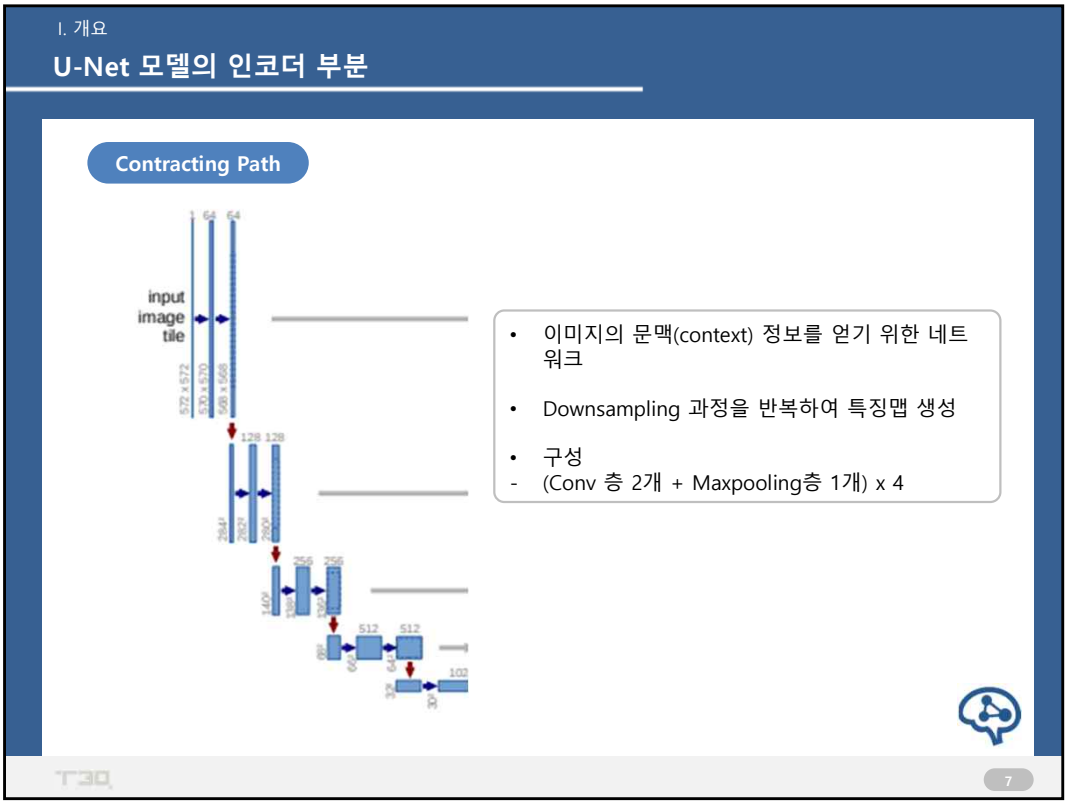


- Medical분야의 MICCAI 2015학회에서 발표된 U-Net: Convolutional Networks for Biomedical Image Segmentation 논문에서 제안

- Skip Architecture : pooling 과정을 거친 이미지는 가지고 있는 정보를 손실하게 되는데, 이러한 정보의 손실을 방지하고자 다른 layer를 거치는 단계를 skip하고 얕은 층의 layer를 깊은 층의 layer로 직접 연결하는 구조를 말한다.

- Contracting Path : 이미지의 문맥(context) 정보를 얻기 위한 네트워크로 Downsampling과정을 반복하여 특징맵을 생성
- Bottle Neck : 수축경로에서 확장경로로 전환되는 경로
- Expanding Path : 수축 경로에서 추출된 의미정보와 Layer에 존재하는 픽셀의 위치정보를 확장경로의 layer와 결합하여 Up-Sampling을 진행하는 경로
- Skip Connection : 일부 레이어를 건너뛰는 연결을 말하며, U-Net모델에서는 인코딩 레이어와 디코딩 레이어의 직접 연결을 가리킨다.

이미지 출처
- U-Net: Convolutional Networks for Biomedical Image Segmentation 논문



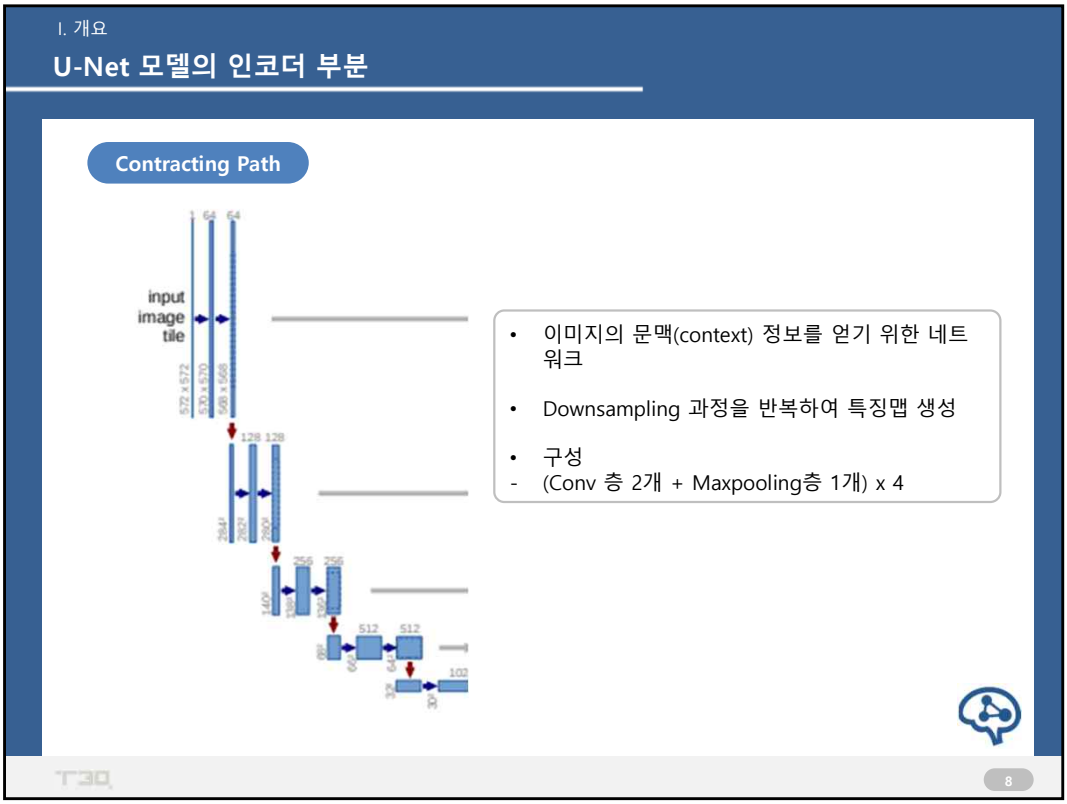
- Contracting Path : 이미지의 문맥(context) 정보를 얻기 위한 네트워크로 Downsampling과정을 반복하여 특징맵을 생성

- U-Net 모델의 인코더 부분
- 각 Contracting step은 2개의 Convolution 층과 Maxpooling 층을 쌓아 4번 반복
- convolution layer층에서는 padding을 진행하지 않으므로 layer 진행 후 해상도 2pixel씩 감소

1번
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(572*572 → 570*570)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(570*570 → 568*568)
- 2×2 Max-polling Layer (Stride 2) 해상도 절반 감소(568*568 → 284*284)

2번
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(284*284 → 282*282), 채널 수 2배 증가(64 → 128)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(282*282 → 280*280)
- 2×2 Max-polling Layer (Stride 2) 해상도 절반 감소(280*280 → 140*140)

이미지 출처
- U-Net: Convolutional Networks for Biomedical Image Segmentation 논문



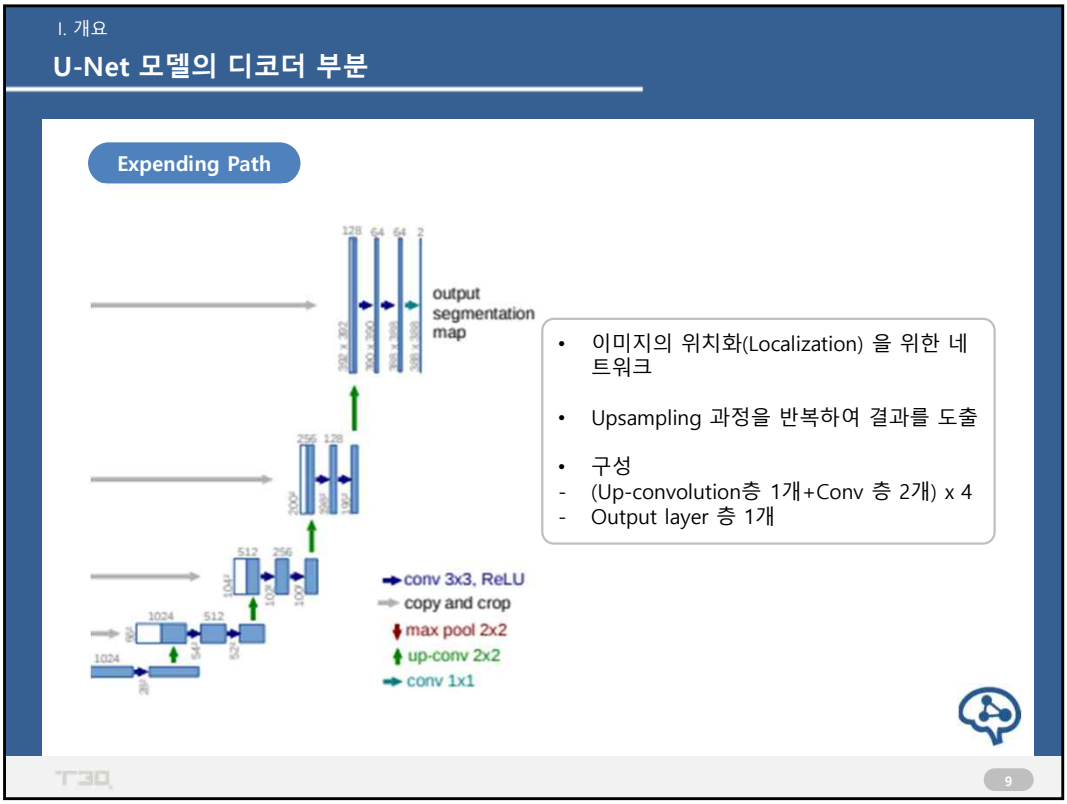
3번
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(140*140 → 138*138), 채널 수 2배 증가(128 → 256)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(138*138 → 136*136)
- 2×2 Max-polling Layer (Stride 2) 해상도 절반 감소(136*136 → 68*68)

4번
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(68*68 → 66*66), 채널 수 2배 증가(256 → 512)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(66*66 → 64*64)
- 2×2 Max-polling Layer (Stride 2) 해상도 절반 감소(64*64 → 32*32)

→ Downsampling 이후 차원과 feature map의 크기는 절반으로 축소, 채널의 수는 2배 증가

- Bottle Neck
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(32*32 → 30*30)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) → 해상도 감소(30*30 → 28*28)

이미지 출처
- U-Net: Convolutional Networks for Biomedical Image Segmentation 논문



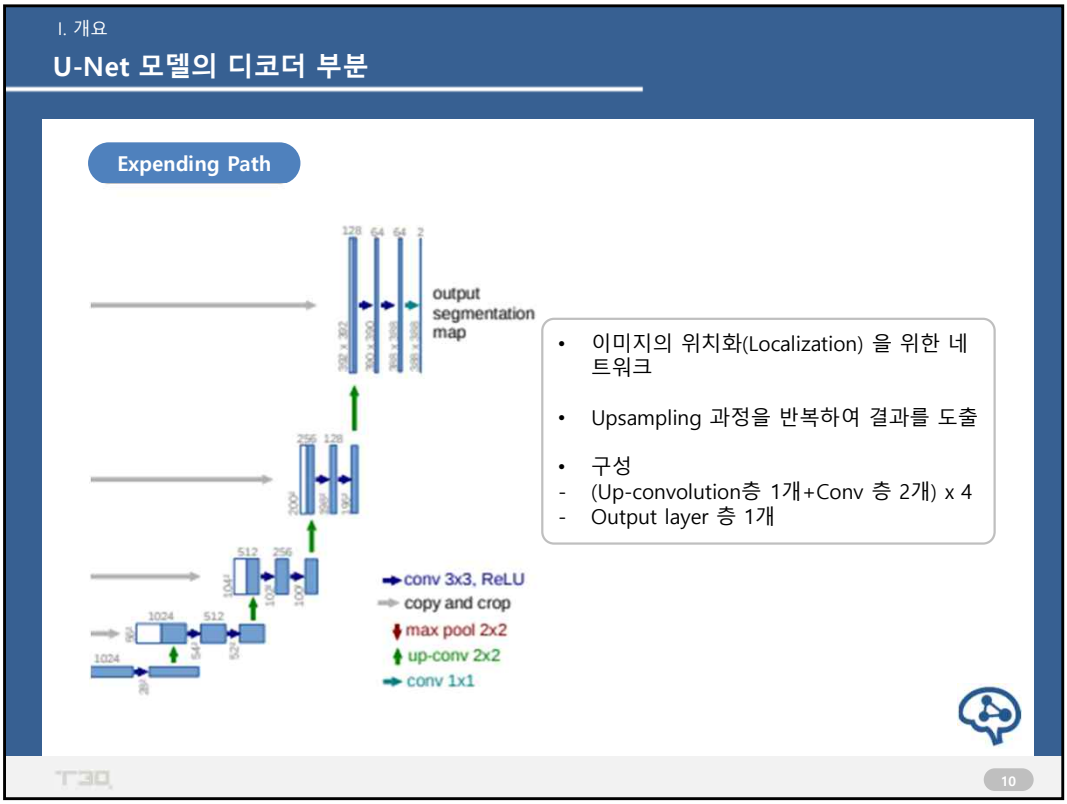
- Expanding Path : Localization을 위한 네트워크로 Upsampling 과정을 반복하여 결과를 도출

- U-Net 모델의 디코더 부분
- 각 Expanding step은 1개의 Up-convolution 층과 2개의 Convolution 층을 쌓아 4번 반복
- convolution layer층에서는 padding을 진행하지 않으므로 layer 진행 후 해상도 2pixel씩 감소

1번
- 2x2 Up-convolution 해상도 2배 증가(28*28 → 56*56)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1), 해상도 감소(56*56 → 54*54), 채널 수 감소(1024 → 512)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1), 해상도 감소(54*54 → 52*52)

2번
- 2x2 Up-convolution 해상도 2배 증가(52*52 → 104*104)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1), 해상도 감소(104*104 → 102*102), 채널 수 감소(512 → 256)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1), 해상도 감소(102*102 → 100*100)

이미지 출처
- U-Net: Convolutional Networks for Biomedical Image Segmentation 논문



3번
- 2x2 Up-convolution 해상도 2배 증가(100*100 → 200*200)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1), 해상도 감소(200*200 → 198*198), 채널 수 감소(256 → 128)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1), 해상도 감소(198*198 → 196*196)

4번
- 2x2 Up-convolution(Conv2DTranspose) 해상도 2배 증가(196*196 → 392*392)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1), 해상도 감소(392*392 → 390*390), 채널 수 감소(128 → 64)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1), 해상도 감소(390*390 → 388*388)

→ 4번 반복할 때 각 단계는 수축경로에서 추출된 feature map을 잘라내어 skip 연결로 얕은 층의 layer와 깊은 층의 layer를 결합하여 학습 진행

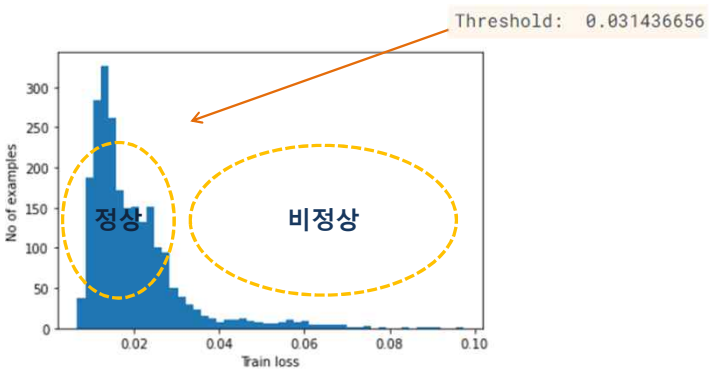
- output layer : 1*1 Convolution Layer 결과출력

이미지 출처
- U-Net: Convolutional Networks for Biomedical Image Segmentation 논문

I. 개요

추론

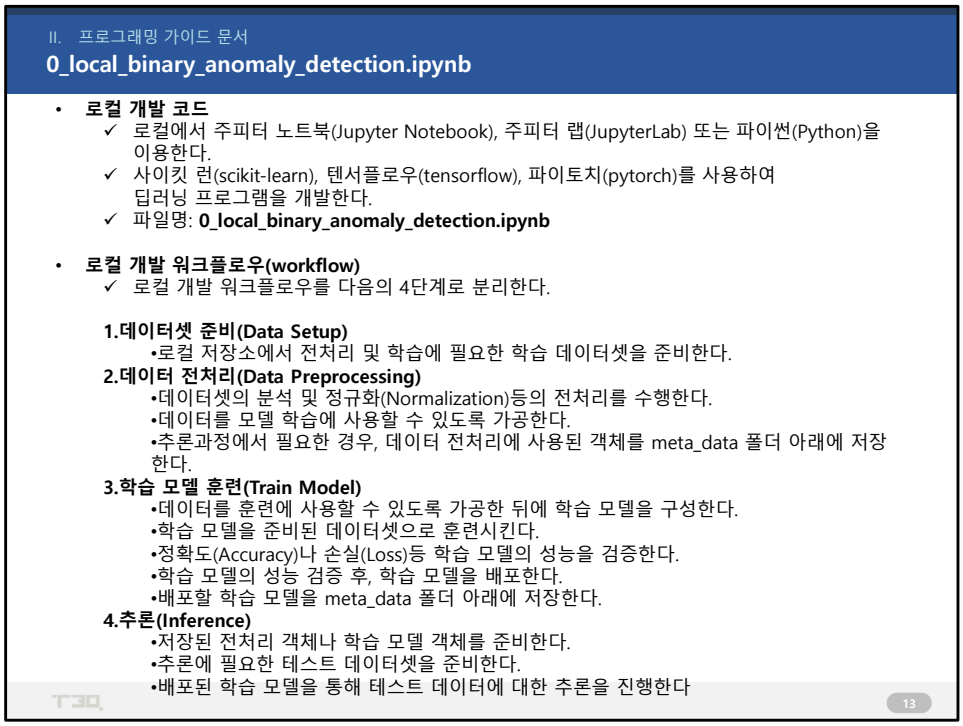
- 오토인코더를 이용한 이상탐지의 경우 Threshold 를 기준으로 이상데이터를 탐지
- Threshold : 이상탐지의 기준이 되는 임계값
- 해당 예제에서는 학습 시 도출된 loss값의 평균과 표준편차를 더하여 Threshold를 도출
- Threshold를 기준으로 왼쪽 : 정상데이터
- Threshold를 기준으로 오른쪽 : 비정상데이터



이미지 출처 : https://www.tensorflow.org/tutorials/generative/autoencoder(tensorflow 공식 홈페이지)

0_local_binary_anomaly_detection_requirement

- 로컬에 설치 되어야 할 패키지 버전
  - ✓ numpy 1.21.5
  - ✓ pandas 1.3.5
  - ✓ matplotlib 3.5.2
  - ✓ tensorflow 2.8.0
  - ✓ tensorflow-gpu 2.8.0
  - ✓ keras 2.8.0
  - ✓ PIL 9.0.1



0_local_binary_anomaly_detection.ipynb

```
==========================================
# Imports
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import zipfile
import os
import tensorflow as tf
from tensorflow.keras import layers,losses
from sklearn.utils import shuffle
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Conv2DTranspose
from tensorflow.keras.layers import Activation, BatchNormalization, Concatenate
import cv2
import pickle
```

1. 데이터셋 준비(Data Setup)

```
# dataset.zip 파일을 dataset 폴더에 압축을 풀어준다.
zip_target_path = './meta_data'
zip_source_path = './dataset.zip'

extract_zip_file = zipfile.ZipFile(zip_source_path)
extract_zip_file.extractall(zip_target_path)

extract_zip_file.close()

# 필요한 변수들을 전역변수로 정의
input_shape = (256, 256)
```

```
# 이미지 데이터 로드하기
x_data = tf.keras.preprocessing.image_dataset_from_directory(
    './meta_data/dataset/',
    color_mode='grayscale',
    image_size=input_shape,
    batch_size = 1
)
# 데이터 클래스 확인
x_data.class_name
```

2. 데이터 전처리 (Data Preprocessing)
데이터 생성

```
# benign data : 122개 -> 학습에는 정상데이터만 활용됨
benign_data = []
benign_label = []

# malicious data : 160개 -> 검증데이터로만 사용됨
mal_data = []
mal_label = []

# train_data : benign data 100개 사용
train_data=[]
train_label=[]

# valid_data : benign data 22개, malicious data 22개 사용
valid_data = []
valid_label = []

for image, label in x_data: # x_data에서 데이터를 꺼내어 benign_data와 mal_data로 나눠 담는다
    image = np.reshape(image, (256, 256, 1))
    label = np.array(label)
    # label이 0이면 benign data 아니면 mal data에 추가
    benign_data.append(image) if label==0 else mal_data.append(image)
    benign_label.append(label) if label==0 else mal_label.append(label)

# train data 생성
train_data=benign_data[:100]
train_label=benign_label[:100]

# valid data 생성
for data in benign_data[100:]:
    valid_data.append(data)
for data in mal_data[:22]:
    valid_data.append(data)
# valid label 생성
for label in benign_label[100:]:
    valid_label.append(label)
for label in mal_label[:22]:
    valid_label.append(label)
```

정상데이터 시각화

```
# benign_data 정상 데이터
plt.figure(figsize =(15,5))
```

```
for i in range(3) :
    plt.subplot(1, 3, i + 1)
    plt.imshow(benign_data[i], cmap='gray')
    plt.axis('off')

plt.suptitle('benign data', fontsize=20)
plt.show()

# mal_data 비정상 데이터
plt.figure(figsize =(15,5))

for i in range(3) :
    plt.subplot(1, 3, i + 1)
    plt.imshow(mal_data[i], cmap='gray')
    plt.axis('off')

plt.suptitle('malicious data', fontsize=20)
plt.show()
```

데이터 전처리
학습데이터와 검증데이터의 0-255사이의 픽셀값을 각각 0-1 사이로 normalization해준다.

```
train_x= np.asarray(tf.cast(train_data, tf.float32) / 255.0)
valid_x = np.asarray(tf.cast(valid_data, tf.float32) / 255.0)
valid_y = np.array(valid_label).reshape(-1,)

# shape 확인
train_x.shape, valid_x.shape, valid_y.shape

# 검증 데이터 섞어줌
valid_x, valid_y = shuffle(valid_x,valid_y)
```

3. 학습 모델 훈련(Train Model)
U-Net
- U-Net 모델은 오코인코더 기반으로 입력과 출력이 동일한 값을 가지는 네트워크 구조를 가지며 서로 대칭되는 인코더와 디코더, 둘을 결합하는 층으로 구성된다.
- 저차원과 고차원 정보를 모두 사용하여 이미지의 특징을 추출하는 인공신경망으로 각 인코더레이어와 디코더레이어가 직접연결되는 Skip Architecture 개념이 사용됨
- U-Net 모델의 구조는 중심점을 기준으로 좌우가 대칭되는 U자 형을 이루고 있음

1. Contracting Path -> 이미지의 context 정보를 얻기위한 네트워크로 Downsampling과정을 반복하여 특징맵을 생성
- 인코더
- 각 Contracting step은 2개의 Convolution 층과 Maxpooling 층을 쌓는다.
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1) -> 이미지 특징정보 추출 padding을 하지 않으므로 특징맵의 크기가 감소
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1)
- 2×2 Max-polling Layer (Stride 2) -> 차원축소, 채널의 수를 2배 증가시키므로 채널은 Downsampling 진행시마다 증가
> 4번 반복

2. Bottle Neck -> Contracting Path에서 Expanding Path로 전환되는 구간
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1)

> 1번

3. Expanding Path -> Localization 을 위한 네트워크로 Upsampling 과정을 반복하여 결과를 도출
- 디코더
- 각 Expanding step은 1개의 Up-convolution 층과 2개의 Convolution 층을 쌓는다.
- 2x2 Up-convolution(Conv2DTranspose) (feature map 크기 두배로 증가)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1)
- 3×3 Convolution Layer + ReLu + BatchNorm (No Padding, Stride 1)
> 4번 반복
- output layer : 결과출력

```
# 모델 정의
class ConvBlock(tf.keras.layers.Layer): # convolution + batch_normalization + relu 층 2개로 구성
    def __init__(self, n_filters):
        super(ConvBlock, self).__init__()

        self.conv1 = Conv2D(n_filters, 3, padding='same')
        self.conv2 = Conv2D(n_filters, 3, padding='same')

        self.bn1 = BatchNormalization()
        self.bn2 = BatchNormalization()

        self.activation = Activation('relu')

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.bn1(x)
        x = self.activation(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = self.activation(x)
        return x

class EncoderBlock(tf.keras.layers.Layer):# ConvBlock(convolution 층 2개) + maxpooling
    def __init__(self, n_filters):
        super(EncoderBlock, self).__init__()

        self.conv_blk = ConvBlock(n_filters)
        self.pool = MaxPooling2D((2,2))

    def call(self, inputs):
        x = self.conv_blk(inputs)
        p = self.pool(x)
        return x, p

class DecoderBlock(tf.keras.layers.Layer): # up convolution 층 + ConvBlock(convolution 층 2개)
    def __init__(self, n_filters):
        super(DecoderBlock, self).__init__()

        self.up = Conv2DTranspose(n_filters, (2,2), strides=2, padding='same')
        self.conv_blk = ConvBlock(n_filters)
    def call(self, inputs, skip):
        x = self.up(inputs)
        x = Concatenate()([x, skip])
        x = self.conv_blk(x)
```

```
return x
```

```
class UNET(tf.keras.Model):
    def __init__(self):
        super(UNET, self).__init__()

        # Contracting path
        self.e1 = EncoderBlock(64)
        self.e2 = EncoderBlock(128)
        self.e3 = EncoderBlock(256)
        self.e4 = EncoderBlock(512)

        # Bottle neck
        self.b = ConvBlock(1024)

        # Expending path
        self.d1 = DecoderBlock(512)
        self.d2 = DecoderBlock(256)
        self.d3 = DecoderBlock(128)
        self.d4 = DecoderBlock(64)

        # Outputs
        self.outputs = Conv2D(1, 1, padding='same', activation='sigmoid') #비선형 예측을 위한 1*1
                                                                          #conv연산 추가
    def call(self, inputs):
        s1, p1 = self.e1(inputs)
        s2, p2 = self.e2(p1)
        s3, p3 = self.e3(p2)
        s4, p4 = self.e4(p3)

        b = self.b(p4)

        d1 = self.d1(b, s4)
        d2 = self.d2(d1, s3)
        d3 = self.d3(d2, s2)
        d4 = self.d4(d3, s1)

        outputs = self.outputs(d4)
        return outputs

unet_model = UNET()

# optimizer 설정(learning_rate를 0.1로 설정해준다)
opt_adam = tf.keras.optimizers.Adam(
    learning_rate=0.1,)

unet_model.compile(optimizer=opt_adam, loss=losses.MeanAbsoluteError())

# callback함수 정의
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    verbose=2,
    mode='auto',
    cooldown=0,
```

min_lr=0.0001)

```
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    mode='auto',
    patience=10,
    verbose=0
)

history = unet_model.fit(train_x, train_x,
                         epochs=40,
                         shuffle=True,
                         validation_data= (valid_x, valid_x),
                         callbacks = [early_stop, reduce_lr],
                         batch_size = 2)

unet_model.summary()
```

모델 평가 (Evaluate Model)

```
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
```

악성코드와 정상코드를 탐지할 threshold값 구하기

```
def calculate_loss(train_data, model):
    reconstructions = model.predict(train_data)
    train_loss = abs(np.mean(train_data) - np.mean(reconstructions))
    return train_loss

train_loss = []

for train_data in train_x:
    train_loss.append(calculate_loss(train_data.reshape(-1, 256, 256, 1),unet_model))

# train_loss 의 시각화 및 임계점 확인
plt.hist(train_loss)
plt.xlabel("Train loss")
plt.ylabel("No of examples")
plt.show()

threshold = np.mean(train_loss) + np.std(train_loss)
print("threshold: ", threshold)
```

4. 추론(Inference)

악성코드 감지

- 오토인코더 모델은 정상코드 이미지를 재구축 하도록 훈련되었기 때문에 비악성 코드가 아닌 악성 코드 이미지를 재구성하기는 어렵다.
- 그렇기 때문에 비악성코드 파일의 경우 낮은 재구성 오류율이 발생하고 악성 코드의 경우 높은 재구성 오류율이 발생한다.
- 결과적으로, 악성과 비악성 실행 파일을 분류하는 임계값을 설정하고 임계값보다 오류율이 낮은 실행 파일을 정상 코드로 분류한다.
- 이상탐지 결과: 비악성 코드(True), 악성 코드(False)

악성코드(malware) 데이터 예측

```
# test_dataset 압축해제
zip_target_path = './meta_data'
zip_source_path = './test_dataset.zip'

extract_zip_file = zipfile.ZipFile(zip_source_path)
extract_zip_file.extractall(zip_target_path)

extract_zip_file.close()

# 테스트 이미지 데이터 로드하기
test_data = tf.keras.preprocessing.image_dataset_from_directory(
    './meta_data/test_dataset',

    color_mode='grayscale',
    image_size=input_shape,
    batch_size = 1
)

# test_data class 확인 -> 정상이 0, 비정상이 1로 labeling
test_data.class_names

test_x = []
test_y = []

for image, label in test_data: # test_data에서 데이터를 꺼내어 test_x, test_y에 담아준다.
    image = np.reshape(image, (256, 256, 1))
    label = np.array(label)
    test_x.append(image)
    test_y.append(label)

# 0~255값을 0~1로 normalization
test_x= np.asarray(tf.cast(test_x, tf.float32) / 255.0)
test_y = np.array(test_y).reshape(-1,)

# loss값 계산
test_loss = []
for test_data in test_x:
    test_loss.append(calculate_loss(test_data.reshape(-1, 256, 256, 1),unet_model))
test_loss = np.array(test_loss)

# loss와 threshold를 비교하여 y값을 예측
# tf.math.less(x, y) -> x가 y 보다 작으면 True / loss < threshold = True
pred_y = np.array(tf.math.less(test_loss, threshold))
# 정상치 입력 -> 복원 잘되서 loss < threshold / True = 1로 결과 추출 -> 정상 레이블 값인 0으로
labeling
predict_y = np.where(pred_y,0,1)
actual_y = test_y
```

0 : 비악성코드 / 1 : 악성코드

```
print('predict_y :', predict_y)
print('actucal_y :', actual_y)
print(np.mean(np.equal(actual_y,predict_y)))
```

1_local_platform_binary_anomaly_detection.ipynb

◆ **플랫폼 업로드를 쉽게하기 위한 로컬 개발 코드**

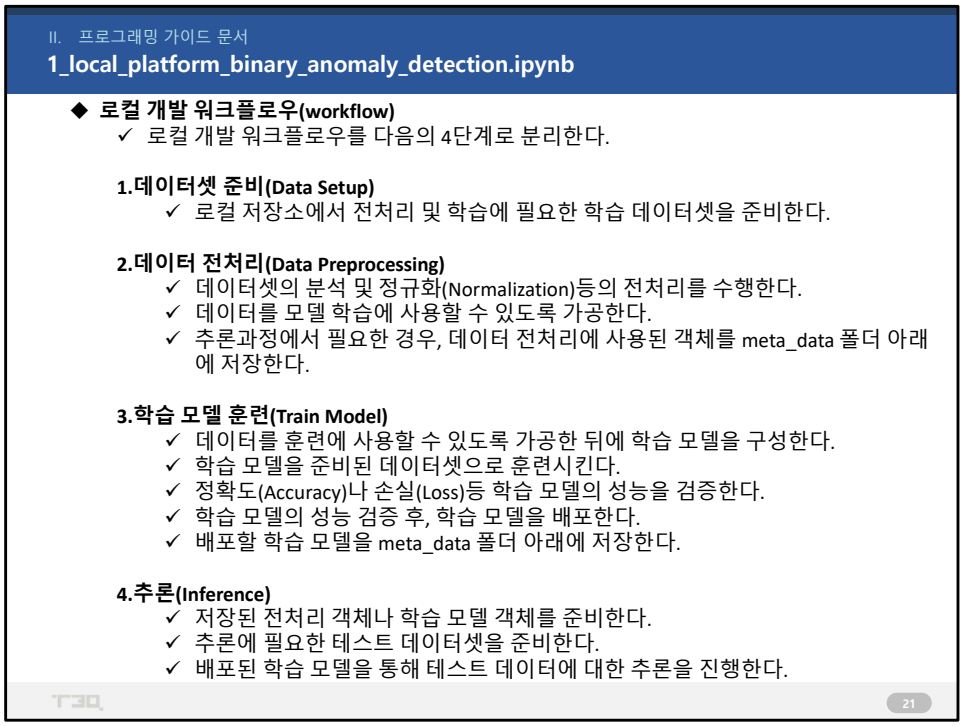
- ✓ T3Q.ai(T3Q.cep + T3Q.dl): 빅데이터/인공지능 통합 플랫폼
- ✓ 플랫폼 업로드를 쉽게하기 위하여 로컬에서 아래의 코드(파일1)를 개발한다.
- ✓ 파일 1(파일명): 1_local_platform_binary_anomaly_detection.ipynb

▪ **전처리 객체 또는 학습모델 객체**

- ✓ 전처리 객체나 학습모델 객체는 meta_data 폴더 아래에 저장한다.

▪ **데이터셋(학습 데이터/테스트 데이터)**

- ✓ 학습과 테스트에 사용되는 데이터를 나누어 관리한다.
- ✓ 학습 데이터: dataset 폴더 아래에 저장하거나 dataset.zip 파일 형태로 저장한다.
- ✓ 테스트 데이터: test_dataset 폴더 아래에 저장하거나 test_dataset.zip 파일 형태로 저장한다.




```
# 파일명: binary_anomaly_detection_preprocess.py

'''
from binary_anomaly_detection_preprocess_sub import exec_process
'''
import logging

logging.basicConfig(level=logging.INFO)

def process_for_train(pm):

    exec_process(pm)

    logging.info('[hunmin log] the end line of the function [process_for_train]')


def init_svc(im, rule):
    return {}


def transform(df, params, batch_id):

    logging.info('[hunmin log] df : {}'.format(df))
    logging.info('[hunmin log] df.shape : {}'.format(df.shape))
    logging.info('[hunmin log] type(df) : {}'.format(type(df)))
    logging.info('[hunmin log] the end line of the function [transform]')

    return df
```

```
# 파일명: binary_anomaly_detection_preprocess_sub.py

import os
import logging
import zipfile

def exec_process(pm):

    logging.info('[hunmin log]  the start line of the function [exec_process]')

    logging.info('[hunmin log] pm.source_path : {}'.format(pm.source_path))

    source_path = pm.source_path
    target_path = pm.target_path

    # 저장 파일 확인
    list_files_directories(source_path)

    my_zip_path = os.path.join(source_path,'dataset.zip')

    extract_zip_file = zipfile.ZipFile(my_zip_path)
    extract_zip_file.extractall(os.path.join(target_path,))

    extract_zip_file.close()

    logging.info('[hunmin log]  the finish line of the function [exec_process]')


# 저장 파일 확인
def list_files_directories(path):
    # Get the list of all files and directories in current working directory
    dir_list = os.listdir(path)

    logging.info('[hunmin log] Files and directories in {} :'.format(path))
    logging.info('[hunmin log] dir_list : {}'.format(dir_list))
```

```
# 파일명: binary_anomaly_detection_train.py

'''
from binary_anomaly_detection_train_sub import exec_train, exec_init_svc, exec_inference
'''
import logging


def train(tm):

    exec_train(tm)
    logging.info('[hunmin log] the end line of the function [train]')


def init_svc(im):

    params = exec_init_svc(im)
    logging.info('[hunmin log] the end line of the function [init_svc]')

    return { **params }


def inference(df, params, batch_id):

    result = exec_inference(df, params, batch_id)
    logging.info('[hunmin log] the end line of the function [inference]')

    return { **result }
```

```
# 파일명: binary_anomaly_detection_train_sub.py

# Imports

import pandas as pd
import numpy as np
import os
from tensorflow import keras
from tensorflow.keras import losses
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Conv2DTranspose
from tensorflow.keras.layers import Activation, BatchNormalization, Concatenate
import logging
import pickle
import io
import base64
from PIL import Image
from sklearn.utils import shuffle

logging.info(f'[hunmin log] tensorflow ver : {tf.__version__}')

# 사용할 gpu 번호를 적는다.
os.environ["CUDA_VISIBLE_DEVICES"]='0,1'

gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_visible_devices(gpus, 'GPU')
        logging.info('[hunmin log] gpu set complete')
        logging.info('[hunmin log] num of gpu: {}'.format(len(gpus)))

    except RuntimeError as e:
        logging.info('[hunmin log] gpu set failed')
        logging.info(e)

# 전역변수 정의
input_shape = (256, 256)

########### 플랫폼 train(tm) 부분의 코드 입니다. ###########
def exec_train(tm) :
    logging.info('[hunmin log] train start')
    train_path = os.path.join(tm.train_data_path,'dataset')
    list_files_directories(train_path)

    ######################################################################
    ## 1. 데이터셋 준비(Data Setup)
    ######################################################################

    # 이미지 데이터 로드하기
    x_data = data_load(train_path)

    # 데이터 생성
    train_data, valid_data, valid_label = generate_data(x_data)
```

```
 ###################################################################
## 2. 데이터 전처리(Data Preprocessing)
 ###################################################################
  # data 0~1 사이로 normalization
  train_x= np.asarray(tf.cast(train_data, tf.float32) / 255.0)
  valid_x = np.asarray(tf.cast(valid_data, tf.float32) / 255.0)
  valid_y = np.array(valid_label).reshape(-1,)

  logging.info('[hunmin log] train_x.shape : {}'.format(train_x.shape))
  logging.info('[hunmin log] valid_x.shape : {}'.format(valid_x.shape))
  logging.info('[hunmin log] valid_y.shape : {}'.format(valid_y.shape))
  logging.info('[hunmin log] valid_y : {}'.format(len(valid_y)))

  # valid_x와 valid_y값 shuffle
  valid_x, valid_y = shuffle(valid_x,valid_y)

  ###################################################################
 ## 3. 학습 모델 훈련(Train Model)
  ###################################################################
  logging.info('[hunmin log] model build and compile')

  # batch size 설정
  batch_size = 2 * len(gpus) if len(gpus) > 0 else 2
  logging.info('[hunmin log] batch_size : {}'.format(batch_size))

  # call back 함수 정의
  reduce_lr, early_stop = train_callback()

  # 단일 gpu 혹은 cpu학습
  if len(gpus) < 2:
      unet_model = UNET()
      opt_adam = tf.keras.optimizers.Adam(learning_rate=0.1,)
      unet_model.compile(optimizer=opt_adam, loss=losses.MeanAbsoluteError())
  # multi-gpu
  else:
      strategy = tf.distribute.MirroredStrategy()
      logging.info('[hunmin log] gpu devices num {}'.format(strategy.num_replicas_in_sync))
      with strategy.scope():
          unet_model = UNET()
          opt_adam = tf.keras.optimizers.Adam(learning_rate=0.1,)
          unet_model.compile(optimizer=opt_adam, loss=losses.MeanAbsoluteError())

  logging.info('[hunmin log] model fit start')


  history = unet_model.fit(train_x, train_x,
                           epochs=40,
                           shuffle=True,
                           validation_data = (valid_x,valid_x),
                           callbacks = [early_stop, reduce_lr],
                           batch_size = batch_size)
  logging.info('[hunmin log] model fit complete')
```

```
    ######################################################################
    ## 학습 모델가중치 저장
    ######################################################################
    model_path = tm.model_path
    unet_model.save_weights(os.path.join(model_path,'unet_weight'))
    logging.info('[hunmin log] model_weight_save')

    # 저장 파일 확인
    list_files_directories(model_path)

    # threshold값 구하기
    train_loss = []

    for train_data in train_x:
        train_loss.append(calculate_loss(train_data.reshape(-1, 256, 256, 1),unet_model))

    logging.info('[hunmin log] train_loss.shape : {}'.format(np.array(train_loss).shape))

    threshold = np.mean(train_loss) + np.std(train_loss)
    logging.info("[hunmin log] threshold:{}".format(threshold))

    # threshold 객체값 저장
    with open(os.path.join(tm.model_path, 'threshold.pickle'), 'wb') as fw:
        pickle.dump(threshold, fw)

    # 저장 파일 확인
    list_files_directories(tm.model_path)

    ######################################################################
    ## 플랫폼 시각화
    ######################################################################
    plot_metrics(tm, history, unet_model, valid_x, valid_y, threshold)

    logging.info('[hunmin log]  the finish line of the function [exec_train]')

########### 플랫폼 init_svc부분의 코드입니다. ###########
def exec_init_svc(im):
    ######################################################################
    ## 학습 모델 준비
    ######################################################################
    # load the model_weights
    # parameter에 model 정의
    model_path = im.model_path
    unet_model = UNET()
    unet_model.load_weights(os.path.join(model_path,'unet_weight'))

    # threshold 객체값 읽기
    with open(os.path.join(im.model_path, 'threshold.pickle'), 'rb') as fr:
        threshold = pickle.load(fr)

    params = {'unet_model' : unet_model, 'threshold':threshold}
    return params
```

```
########### 플랫폼 inference부분의 코드입니다. ###########
def exec_inference(df, params, batch_id) :
    ###################################################################
   ## 4. 추론(Inference)
    ###################################################################
  logging.info('[hunmin log]  the start line of the function [exec_inference]')

    # model load
    unet_model = params['unet_model']

    # threshold 객체값 load
    threshold = params['threshold']

    df = io.BytesIO(base64.b64decode(df.iloc[0, 0]))
    logging.info('[hunmin log] decode_df')

    # image preprocessing
    img = image_trans(df)

    results = {}
    loss = calculate_loss(img,unet_model)
    pred_y = np.array(tf.math.less(loss, threshold))
    predict_y = np.where(pred_y, 0, 1)

    results['inference'] = 'benign' if predict_y.tolist() == 0 else 'malware'

    logging.info('[hunmin log] results : {}'.format(results))
    logging.info('[hunmin log]  the finish line of the function [exec_inference]')

    return results

#########################################################################
## exec_train(tm) 호출 함수
#########################################################################
# 저장 파일 확인
def list_files_directories(path):
    # Get the list of all files and directories in current working directory
    dir_list = os.listdir(path)
    logging.info('[hunmin log] Files and directories in {} :'.format(path))
    logging.info('[hunmin log] dir_list : {}'.format(dir_list))

# 데이터 로드
def data_load(train_path) :
    x_data = tf.keras.preprocessing.image_dataset_from_directory(
        train_path,
        color_mode='grayscale',
        image_size=input_shape,
        batch_size = 1
    )
    logging.info('[hunmin log] x_data.class_names :{}'.format(x_data.class_names))
    return x_data
```

```
# 데이터 생성
def generate_data(x_data) :

    benign_data = []
    benign_label = []

    mal_data = []
    mal_label = []

    valid_data = []
    valid_label = []

    for image, label in x_data: # benign_data에 x_data에서 image를 꺼내어 담아준다.
        image = np.reshape(image, (256, 256, 1))
        label = np.array(label)
        # label이 0이면 benign data 아니면 mal data에 추가
        benign_data.append(image) if label==0 else mal_data.append(image)
        benign_label.append(label) if label==0 else mal_label.append(label)
    logging.info('[hunmin log] benign_data len : {}'.format(len(benign_data)))
    logging.info('[hunmin log] mal_data len : {}'.format(len(mal_data)))

     # train data 정의
    train_data=benign_data[:100]
    train_label=benign_label[:100]

    # valid data 생성
    for data in benign_data[100:]:
        valid_data.append(data)
    for data in mal_data[:22]:
        valid_data.append(data)
    # valid label 생성
    for label in benign_label[100:]:
        valid_label.append(label)
    for label in mal_label[:22]:
        valid_label.append(label)
    logging.info('[hunmin log] valid_data len : {}'.format(len(valid_data)))
    logging.info('[hunmin log] valid_label len : {}'.format(len(valid_label)))
    return train_data, valid_data, valid_label

# call back 함수 정의
def train_callback():
    reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
                monitor='val_loss',
                factor=0.5,
                patience=5,
                verbose=0,
                mode='auto',
                cooldown=0,
                min_lr=0.0001
                )
    early_stop = tf.keras.callbacks.EarlyStopping(
                    monitor='val_loss',
                    mode='auto',
                    patience=10
                )
    return reduce_lr, early_stop
```

```
# loss 계산
def calculate_loss(train_data, model):
    reconstructions = model.predict(train_data)
    train_loss = abs(np.mean(train_data) - np.mean(reconstructions))
    return train_loss

# 시각화
def plot_metrics(tm, history, model, x_test, y_test, threshold):
    from sklearn.metrics import accuracy_score, confusion_matrix, log_loss

    test_loss = []
    for test_data in x_test:
        test_loss.append(calculate_loss(test_data.reshape(-1, 256, 256, 1),model))

    test_loss = np.array(test_loss)

    pred_y = np.array(tf.math.less(test_loss, threshold))
    #tf.math.less(x, y) -> x가 y 보다 작으면 True / loss< threshold True
    #) 정상치 입력 -> 복원 잘되서 loss가 threshold보다 작음 True = 1 -> 0으로 labeling
    # label 0이 benign data

    predict_y =np.where(pred_y, 0, 1)
    actual_y = y_test

    predict_y = predict_y.tolist()
    actual_y = actual_y.tolist()

    logging.info('[hunmin log] predict_y : {}'.format(predict_y))
    logging.info('[hunmin log] actual_y : {}'.format(actual_y))

    loss_list = history.history['loss']

    # train의 accuracy 대신 validation의 accuracy 출력
    test_accuracy =np.mean(np.equal(actual_y,predict_y))

    for step, loss in enumerate(loss_list):
        metric={}
        metric['accuracy'] = test_accuracy
        metric['loss'] = loss
        metric['step'] = step
#         tm.save_stat_metrics(metric)

    eval_results={}
    eval_results['predict_y'] = predict_y
    eval_results['actual_y'] = actual_y
    eval_results['accuracy'] = test_accuracy
    eval_results['loss'] = history.history['loss'][-1]
    logging.info('[hunmin log] accuracy{}'.format(eval_results['accuracy']))

    # calculate_confusion_matrix(eval_results)
    eval_results['confusion_matrix'] = confusion_matrix(actual_y, predict_y).tolist()
#     tm.save_result_metrics(eval_results)
    logging.info('[hunmin log] accuracy and loss curve plot for platform')
```

```
# 모델 정의
class ConvBlock(tf.keras.layers.Layer): # convolution + batch_normalization + relu 층 2개로 구성
    def __init__(self, n_filters):
        super(ConvBlock, self).__init__()

        self.conv1 = Conv2D(n_filters, 3, padding='same')
        self.conv2 = Conv2D(n_filters, 3, padding='same')

        self.bn1 = BatchNormalization()
        self.bn2 = BatchNormalization()

        self.activation = Activation('relu')

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.bn1(x)
        x = self.activation(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = self.activation(x)

        return x


class EncoderBlock(tf.keras.layers.Layer):# ConvBlock(convolution 층 2개) + maxpooling
    def __init__(self, n_filters):
        super(EncoderBlock, self).__init__()

        self.conv_blk = ConvBlock(n_filters)
        self.pool = MaxPooling2D((2,2))

    def call(self, inputs):
        x = self.conv_blk(inputs)
        p = self.pool(x)
        return x, p


class DecoderBlock(tf.keras.layers.Layer): # up convolution 층 + ConvBlock(convolution 층 2개)
    def __init__(self, n_filters):
        super(DecoderBlock, self).__init__()

        self.up = Conv2DTranspose(n_filters, (2,2), strides=2, padding='same')
        self.conv_blk = ConvBlock(n_filters)

    def call(self, inputs, skip):
        x = self.up(inputs)
        x = Concatenate()([x, skip])
        x = self.conv_blk(x)

        return x
```

```
class UNET(tf.keras.Model):
    def __init__(self):
        super(UNET, self).__init__()

        # Contracting path
        self.e1 = EncoderBlock(64)
        self.e2 = EncoderBlock(128)
        self.e3 = EncoderBlock(256)
        self.e4 = EncoderBlock(512)

        # Bottle neck
        self.b = ConvBlock(1024)

        # Expending path
        self.d1 = DecoderBlock(512)
        self.d2 = DecoderBlock(256)
        self.d3 = DecoderBlock(128)
        self.d4 = DecoderBlock(64)

        # Outputs       #비선형 예측을 위한 1*1 conv연산 추가
        self.outputs = Conv2D(1, 1, padding='same', activation='sigmoid')

    def call(self, inputs):
        s1, p1 = self.e1(inputs)
        s2, p2 = self.e2(p1)
        s3, p3 = self.e3(p2)
        s4, p4 = self.e4(p3)

        b = self.b(p4)

        d1 = self.d1(b, s4)
        d2 = self.d2(d1, s3)
        d3 = self.d3(d2, s2)
        d4 = self.d4(d3, s1)

        outputs = self.outputs(d4)

        return outputs


########################################################################
## exec_inference(df, params, batch_id) 호출 함수
########################################################################
# 이미지 변환
def image_trans(df) :
    image = Image.open(df)
    image = image.resize((256,256))
    image = image.convert('L')
    image = np.array(image)
    image = np.asarray(tf.cast(image,tf.float32)/255.0)
    logging.info('[hunmin log] image_shape : {}'.format(image.shape))
    image = np.array(image).reshape(1,256,256,1)
    return image
```

```
# PM 클래스: pm 객체
class PM:
    def __init__(self):
        self.source_path = './'
        self.target_path = './meta_data'

# TM 클래스: tm 객체
class TM:
    param_info = {}
    def __init__(self):
        self.train_data_path = './meta_data'
        self.model_path = './meta_data'
        self.param_info['batch_size'] = 10
        self.param_info['epoch'] = 20

# IM 클래스: im 객체
class IM:
    def __init__(self):
        self.model_path = './meta_data'

# pm 객체
pm = PM()
print('pm.source_path:', pm.source_path)
print('pm.target_path: ', pm.target_path)

# tm 객체
tm = TM()
print('tm.train_data_path: ', tm.train_data_path)
print('tm.model_path: ', tm.model_path)
print('tm.param_info[₩'batch_size₩']: ', tm.param_info['batch_size'])
print('tm.param_info[₩'epoch₩']: ', tm.param_info['epoch'])

# im 객체
im = IM()
print('im.model_path: ', im.model_path)

# inferecne(df, params, batch_id) 함수 입력
params = {}
batch_id = 0

import io
import pandas as pd

# base64 encoded image
data = # '(11_1) Request_binary_anomaly_detection.txt' 파일 안에 있는 base64 형식의 데이터 입력
df = pd.DataFrame(data)
print('df: ', df)
print('df.dtypes:', df.dtypes)
df.columns
```

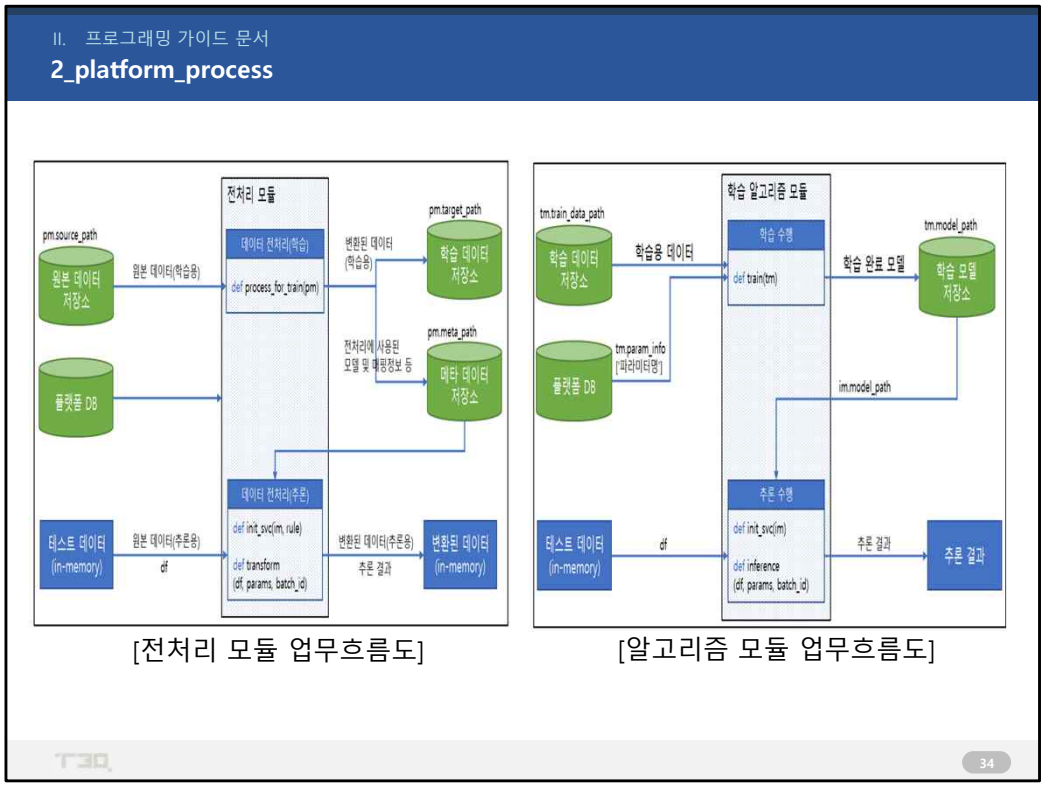
2_platform_process

- 파일명 : binary_anomaly_detection_preprocess.py
  - ✓ from binary_anomaly_detection_preprocess_sub import exec_process
  - ✓ def process_for_train(pm):
  - ✓ def init_svc(im, rule):
  - ✓ def transform(df, params, batch_id):

- 파일명: binary_anomaly_detection_preprocess_sub.py
  - ✓ def exec_process(pm):

- 파일명: binary_anomaly_detection_train.py
  - ✓ from binary_anomaly_detection_train_sub import exec_train, exec_init_svc, exec_inference
  - ✓ def train(tm):
  - ✓ def init_svc(im):
  - ✓ def inference(df, params, batch_id):

- 파일명binary_anomaly_detection_train_sub.py
  - ✓ def exec_train(tm):
  - ✓ def exec_init_svc(im):
  - ✓ def exec_inference(df, params, batch_id):
    1. 데이터셋 준비(Data Setup)
    2. 데이터 전처리(Data Preprocessing)
    3. 학습 모델 훈련(Train Model)
    4. 추론(Inference)



[전처리 모듈 업무흐름도] [알고리즘 모듈 업무흐름도]

0. 빅데이터/인공지능 통합 플랫폼 [T3Q.ai]
   - 빅데이터 플랫폼 [T3Q.cep]
   - 인공지능 플랫폼 [T3Q.dl]
   - 빅데이터/인공지능 통합 플랫폼 [T3Q.ai (T3Q.cep + T3Q.dl)]

1. 머신러닝(Machine Learning)과 딥러닝(Deep Learning) 프로그래밍 패턴
   (1) 데이터셋 불러오기(Dataset Loading)
   (2) 데이터 전처리(Data Preprocessing)
      - 데이터 정규화(Normalization)
      - 학습과 테스트 데이터 분할(Train/Test Data Split) 등
   (3) 학습 모델 구성(Train Model Build)
   (4) 학습(Model Training)
   (5) 학습 모델 성능 검증(Model Performance Validation)
   (6) 학습 모델 저장(배포) 하기(Model Save)
   (7) 추론 데이터 전처리((Data Preprocessing)
   (8) 추론(Inference) 또는 예측(Prediction)
   (9) 추론 결과 데이터 후처리(Data Postprocessing)

2. 빅데이터/인공지능 통합 플랫폼[T3Q.ai]에서 딥러닝 프로그래밍 하고 인공지능 서비스 실시간 운용하기
   - 6개의 함수로 딥러닝 프로그래밍 하고 인공지능 서비스 실시간 운용하기

   (1) process_for_train(pm) 함수
      - 데이터셋 준비(Dataset Setup)에 필요한 코드 작성
   (2) init_svc(im, rule) 함수
      - 전처리 객체 불러오기 에 필요한 코드 작성(생략 가능)
   (3) transform(df, params, batch_id) 함수

- 추론 데이터 전처리((Data Preprocessing) 에 필요한 코드 작성(생략 가능)

(4) train(tm) 함수
- 데이터셋 불러오기(Dataset Loading)
- 데이터 전처리(Data Preprocessing)
- 학습 모델 구성(Train Model Build)
- 학습(Model Training)
- 학습 모델 성능 검증(Model Performance Validation)
- 전처리 객체 저장
- 학습 모델 저장(배포) 하기에 필요한 코드 작성

(5) init_svc(im) 함수
- 전처리 객체 불러오기
- 학습모델 객체 불러오기에 필요한 코드 작성

(6) inference(df, params, batch_id) 함수
- 추론 데이터 전처리((Data Preprocessing)
- 추론(Inference) 또는 예측(Prediction)
- 추론 결과 데이터 후처리(Data Postprocessing)에 필요한 코드 작성

==========

3. 전처리 모듈 관리, 학습 알고리즘 관리 함수 설명

1) 프로젝트 설정/전처리모듈 관리 함수

```
def process_for_train(pm):
    """
    (1) 입력: pm
      # pm.source_path: 학습플랫폼/데이터셋 관리 메뉴에서 저장한 데이터를 불러오는 경로
      # pm.target_path: 처리 완료된 데이터를 저장하는 경로
    (2) 출력: None
    (3) 설명:
      # 데이터셋 관리 메뉴에서 저장한 데이터를 불러와서 필요한 처리를 수행
      # 처리 완료된 데이터를 저장하는 기능, pm.target_path에 저장
      # train(tm) 함수의 tm.train_data_path를 통해 데이터를 불러와서 전처리와 학습을 수행
    """

def init_svc(im, rule):
    """
    (1) 입력: im, rule
    (2) 출력: None
    (3) 설명:
      # process_for_train(pm) 함수에서 저장한 전처리 객체와 데이터에 적용된 룰(rule)을
        불러오는 기능
      # 전처리 객체, 룰(rule) 불러오기 기능 없이 처리
    """
    return {}

def transform(df, params, batch_id):
    """
    (1) 입력: df, params, batch_id
      # df: 추론모델관리와 추론API관리, 실시간 추론을 통해 전달되는 추론 입력 데이터
        (dataframe 형태)
      # params: init_svc(im, rule) 함수의 리턴(return) 값을 params 변수로 전달
    (2) 출력: df
    (3) 설명:
      # df(추론 입력 데이터)에 대한 전처리를 수행한 후 전처리 된 데이터를
        inference(df, ...) 함수의 입력 df에 전달하는 기능
      # df(추론 입력 데이터)를 전처리 없이 inference(df, params, batch_id) 함수의 입력 df
        에 리턴(return)
    """
    return df
```

2) 프로젝트 설정/학습 알고리즘 관리 함수

```
def train(tm):
    """
    (1) 입력: tm
      # tm.train_data_path: pm.target_path에 저장한 데이터를 불러오는 경로
      # tm.model_path: 전처리 객체와 학습 모델 객체를 저장하는 경로
    (2) 출력: None
    (3) 설명:
      # pm.target_path에 저장한 데이터를 tm.train_data_path를 통해 데이터를 불러오는 기능
      # 데이터 전처리와 학습 모델을 구성하고 모델 학습을 수행
      # 학습 모델의 성능을 검증하고 배포할 학습 모델을 저장
      # 전처리 객체와 학습 모델 객체를 저장, tm.model_path에 저장
      # init_svc(im) 함수의 im.model_path를 통해 전처리 객체와 학습 모델 객체를 준비
    """

def init_svc(im):
    """
    (1) 입력: im
      # im.model_path: tm.model_path에 저장한 전처리 객체와 학습 모델 객체 등을
        불러오는 경로
    (2) 출력: 전처리 객체와 학습 모델 객체 등을 딕셔너리(dictionary) 형태로 리턴(return)
    (3) 설명:
      # tm.model_path에 저장한 전처리 객체와 학습 모델 객체 등을 불러오는 기능
      # 전처리 객체, 룰(rule) 불러오기 기능 없이 처리
      # 전처리 객체와 학습 모델 객체 등을 딕셔너리(dictionary) 형태로 리턴(return)
      # 리턴(return) 값을 inference(df, params, batch_id) 함수의 입력 params 변수로 전달
    """

    return { "model": model, "param": param }

def inference(df, params, batch_id):
    """
    (1) 입력: df, params, batch_id
      # df: transform(df, params, batch_id)함수의 리턴(return) 값으로 전달된 df, 추론 입력
        데이터 (dataframe 형태)
      # params  init_svc(im) 함수의 return 값을 params 변수로 전달
        ## 학습 모델 객체 사용 예시        model=params['model']
        ## 전처리(pca) 객체 사용 예시      pca=params['pca']
    (2) 출력: 추론 결과를 딕셔너리(dictionary) 형태로 리턴(return)
    (3) 설명:
      # 전처리 객체를 사용하여 df(추론 입력 데이터)에 대한 전처리 수행
      # 배포된 학습 모델(model)을 사용하여 df(추론 입력 데이터)에 추론(예측)을 수행
      # 추론 결과를 딕셔너리(dictionary) 형태로 리턴(return)
    """
    return {"inference": result}
```

==========

4. 전처리 모듈 관리, 학습 알고리즘 관리 함수 설명(AI 훈민정음 프로젝트)

1) 프로젝트 설정/전처리모듈 관리 함수(AI 훈민정음 프로젝트)

```
import logging

def process_for_train(pm):
    """
    (1) 입력: pm
        # pm.source_path: 학습플랫폼/데이터셋 관리 메뉴에서 저장한 데이터를 불러오는 경로
        # pm.target_path: 처리 완료된 데이터를 저장하는 경로
    (2) 출력: None
    (3) 설명:
        # 데이터셋 관리 메뉴에서 저장한 데이터를 불러와서 필요한 처리를 수행
        # 처리 완료된 데이터를 저장하는 기능, pm.target_path에 저장
        # train(tm) 함수의 tm.train_data_path를 통해 데이터를 불러와서 전처리와 학습을 수행
    (4) 추가 설명:
        # 함수 구조는 원형대로 유지
        # 실질적인 기능을 하는 함수를 서브모듈 함수(exec_process)로 정의하여 사용함
        # 함수명                          서브함수명
        # process_for_train(pm)          exec_process(pm)
        # 함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행
    """

    exec_process(pm)

    logging.info('[hunmin log] the end line of the function [process_for_train]')

def init_svc(im, rule):
    """
    (1) 입력: im, rule
    (2) 출력: None
    (3) 설명:
      # process_for_train(pm) 함수에서 저장한 전처리 객체와 데이터에 적용된 룰(rule)을
        불러오는 기능
      # 전처리 객체, 룰(rule) 불러오기 기능 없이 처리
    """
    return {}

def transform(df, params, batch_id):
    """
    (1) 입력: df, params, batch_id
      # df: 추론모델관리와 추론API관리, 실시간 추론을 통해 전달되는 추론 입력 데이터
        (dataframe 형태)
      # params: init_svc(im, rule) 함수의 리턴(return) 값을 params 변수로 전달
    (2) 출력: df
    (3) 설명:
      # df(추론 입력 데이터)에 대한 전처리를 수행한 후 전처리 된 데이터를 inference(df, ...)
        함수의 입력 df에 전달하는 기능
      # df(추론 입력 데이터)를 전처리 없이 inference(df, params, batch_id) 함수의 입력 df에
        리턴(return)
    (4) 추가 설명:
      # 함수 구조는 원형대로 유지
      # 함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행
    """
    logging.info('[hunmin log] the end line of the function [transform]')
    return df
```

2) 프로젝트 설정/ 학습 알고리즘 관리 함수(AI 훈민정음 프로젝트)

```
import logging

def train(tm):
    """
    (1) 입력: tm
        # tm.train_data_path: pm.target_path에 저장한 데이터를 불러오는 경로
        # tm.model_path: 전처리 객체와 학습 모델 객체를 저장하는 경로
    (2) 출력: None
    (3) 설명:
        # pm.target_path에 저장한 데이터를 tm.train_data_path를 통해 데이터를 불러오는 기능
        # 데이터 전처리와 학습 모델을 구성하고 모델 학습을 수행
        # 학습 모델의 성능을 검증하고 배포할 학습 모델을 저장
        # 전처리 객체와 학습 모델 객체를 저장, tm.model_path에 저장
        # init_svc(im) 함수의 im.model_path를 통해 전처리 객체와 학습 모델 객체를 준비
    (4) 추가 설명:
        # 함수 구조는 원형대로 유지
        # 실질적인 기능을 하는 함수를 서브모듈 함수(exec_train)로 정의하여 사용함
        # 함수명                          서브함수명
        # train(tm)                       exec_train(tm)
        # 함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행
    """

    exec_train(pm)

    logging.info('[hunmin log] the end line of the function [train]')

def init_svc(im):
    """
    (1) 입력: im
        # im.model_path: tm.model_path에 저장한 전처리 객체와 학습 모델 객체 등을 불러오는
          경로
    (2) 출력: 전처리 객체와 학습 모델 객체 등을 딕셔너리(dictionary) 형태로 리턴(return)
    (3) 설명:
        # tm.model_path에 저장한 전처리 객체와 학습 모델 객체 등을 불러오는 기능
        # 전처리 객체, 룰(rule) 불러오기 기능 없이 처리
        # 전처리 객체와 학습 모델 객체 등을 딕셔너리(dictionary) 형태로 리턴(return)
        # 리턴(return) 값을 inference(df, params, batch_id) 함수의 입력 params 변수로 전달
    (4) 추가 설명:
        # 함수 구조는 원형대로 유지
        # 실질적인 기능을 하는 함수를 서브모듈 함수(exec_init_svc)로 정의하여 사용함
        # 함수명                          서브함수명
        # init_svc(im)                    exec_init_svc(im)
        # 함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행

    """
    params = exec_init_svc(im)
    logging.info('[hunmin log] the end line of the function [init_svc]')
    return {**params}
```

```
def inference(df, params, batch_id):
        """
        (1) 입력: df, params, batch_id
          # df: transform(df, params, batch_id)함수의 리턴(return) 값으로 전달된 df, 추론 입력
            데이터(dataframe 형태)
          # params: init_svc(im) 함수의 리턴(return) 값을 params 변수로 전달
            ## 학습 모델 객체 사용 예시        model=params['model']
            ## 전처리(pca) 객체 사용 예시      pca=params['pca']
        (2) 출력: 추론 결과를 딕셔너리(dictionary) 형태로 리턴(return)
        (3) 설명:
          # 전처리 객체를 사용하여 df(추론 입력 데이터)에 대한 전처리 수행
          # 배포된 학습 모델(model)을 사용하여 df(추론 입력 데이터)에 추론(예측)을 수행
          # 추론 결과를 딕셔너리(dictionary) 형태로 리턴(return)
        (4) 추가 설명:
          # 함수 구조는 원형대로 유지
          # 실질적인 기능을 하는 함수를 서브모듈 함수(exec_inference)로 정의하여 사용함
          # 함수명                                    서브함수명
          # inference(df, params, batch_id)         exec_inference(df, params, batch_id)
          # 함수의 정상적인 동작 체크를 위해 마지막 라인(the end line)에 로그 출력 수행
        """

        result = exec_inference(df, params, batch_id)
        logging.info('[hunmin log] the end line     of the function [inference]')
        return {**result}
```

수행절차 소개

01) T3Q.cep_데이터수집 파이프라인_binary_anomaly_detection : 해당 예제에서는 수행 절차 없음

02) T3Q.cep_데이터변환 파이프라인_binary_anomaly_detection : 해당 예제에서는 수행 절차 없음

03) T3Q.dl_프로젝트 설정_실행환경 관리_binary_anomaly_detection

04) T3Q.dl_프로젝트 설정_전처리 모듈 관리_binary_anomaly_detection

05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_binary_anomaly_detection

06) T3Q.dl_학습플랫폼_데이터셋 관리_binary_anomaly_detection

07) T3Q.dl_학습플랫폼_전처리 모델 설계_binary_anomaly_detection

08) T3Q.dl_학습플랫폼_전처리 모델 관리_binary_anomaly_detection

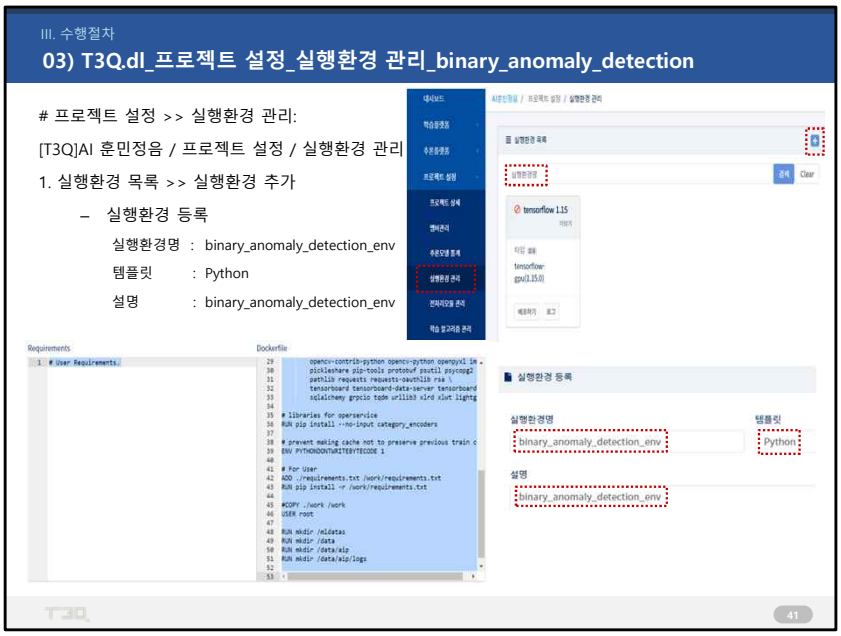
09) T3Q.dl_학습플랫폼_학습모델 설계_binary_anomaly_detection

10) T3Q.dl_학습플랫폼_학습모델 관리_binary_anomaly_detection

11) T3Q.dl_추론플랫폼_추론모델 관리_binary_anomaly_detection

12) T3Q.dl_추론플랫폼_추론API관리_binary_anomaly_detection

13) T3Q.cep_실시간 추론 파이프라인_binary_anomaly_detection



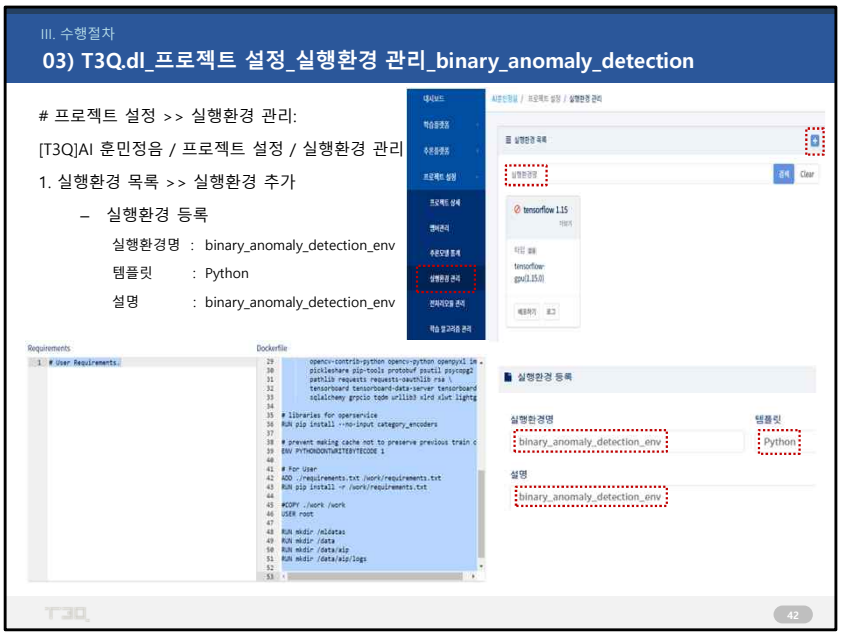
실행환경 추가 내용 및 절차

1) Requirements

```
==================================================
# User Requirements.
==================================================
```

2) Dockerfile

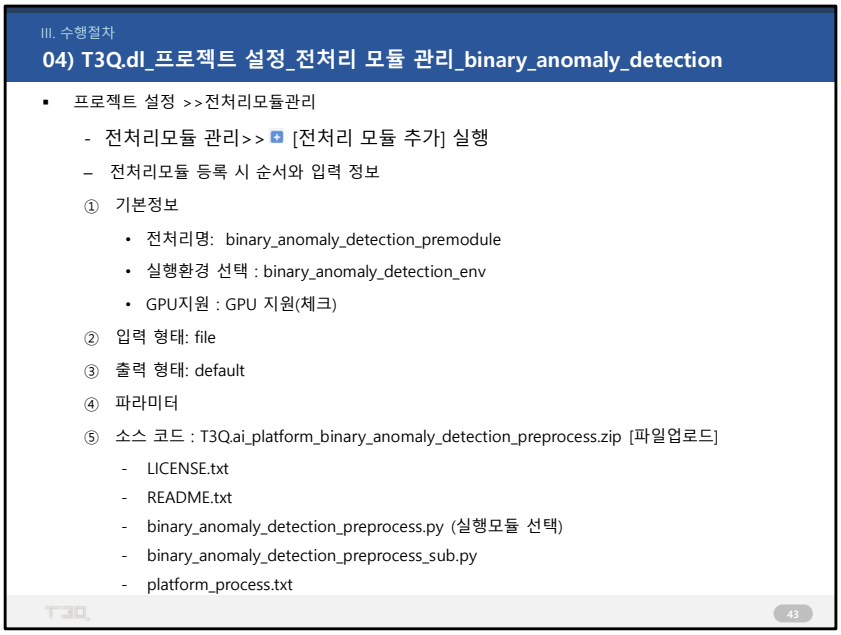
```
==================================================
FROM tensorflow/tensorflow:2.4.1-gpu

ARG DEBIAN_FRONTEND=noninteractive

RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys A4B469963BF863CC

RUN apt-get update && apt-get install -y wget ₩
    python3.8 ₩
    python3-pip ₩
    python3-dev ₩
    python3.8-dev ₩
    postgresql ₩
    libpq-dev

RUN pip3 install --upgrade pip

# libraries for operservice
RUN pip install --no-input kubernetes pygresql pyjwt pyarrow pandas ₩
        flask flask-sqlalchemy flask-cors flask-bcrypt flask-migrate flask-restful flask-rest-
jsonapi

# opencv
RUN apt-get -y install libgl1-mesa-glx
```



==========
2) Dockerfile-계속
==========

```
# generic libraries
RUN pip install --no-input numpy==1.19.5 ₩
        torch scikit-learn imbalanced-learn xgboost ₩
        fastai keras keras-preprocessing keras-vis ₩
        matplotlib pillow nltk ₩
        opencv-contrib-python opencv-python openpyxl imageio pretty_midi ₩
        pickleshare pip-tools protobuf psutil psycopg2 PyYAML ₩
        pathlib requests requests-oauthlib rsa ₩
        tensorboard tensorboard-data-server tensorboard-plugin-wit ₩
        sqlalchemy grpcio tqdm urllib3 xlrd xlwt lightgbm

# libraries for operservice
RUN pip install --no-input category_encoders

# prevent making cache not to preserve previous train code
ENV PYTHONDONTWRITEBYTECODE 1

# For User
ADD ./requirements.txt /work/requirements.txt
RUN pip install -r /work/requirements.txt

#COPY ./work /work
USER root

RUN mkdir /mldatas
RUN mkdir /data
RUN mkdir /data/aip
RUN mkdir /data/aip/logs

WORKDIR/work
```

==========
추가된 실행 환경 [저장] 하고, [배포하기] 시작 , 완료 후 [로그] 에서 확인



- 전처리모듈 등록 시 순서와 입력 정보

① 기본정보

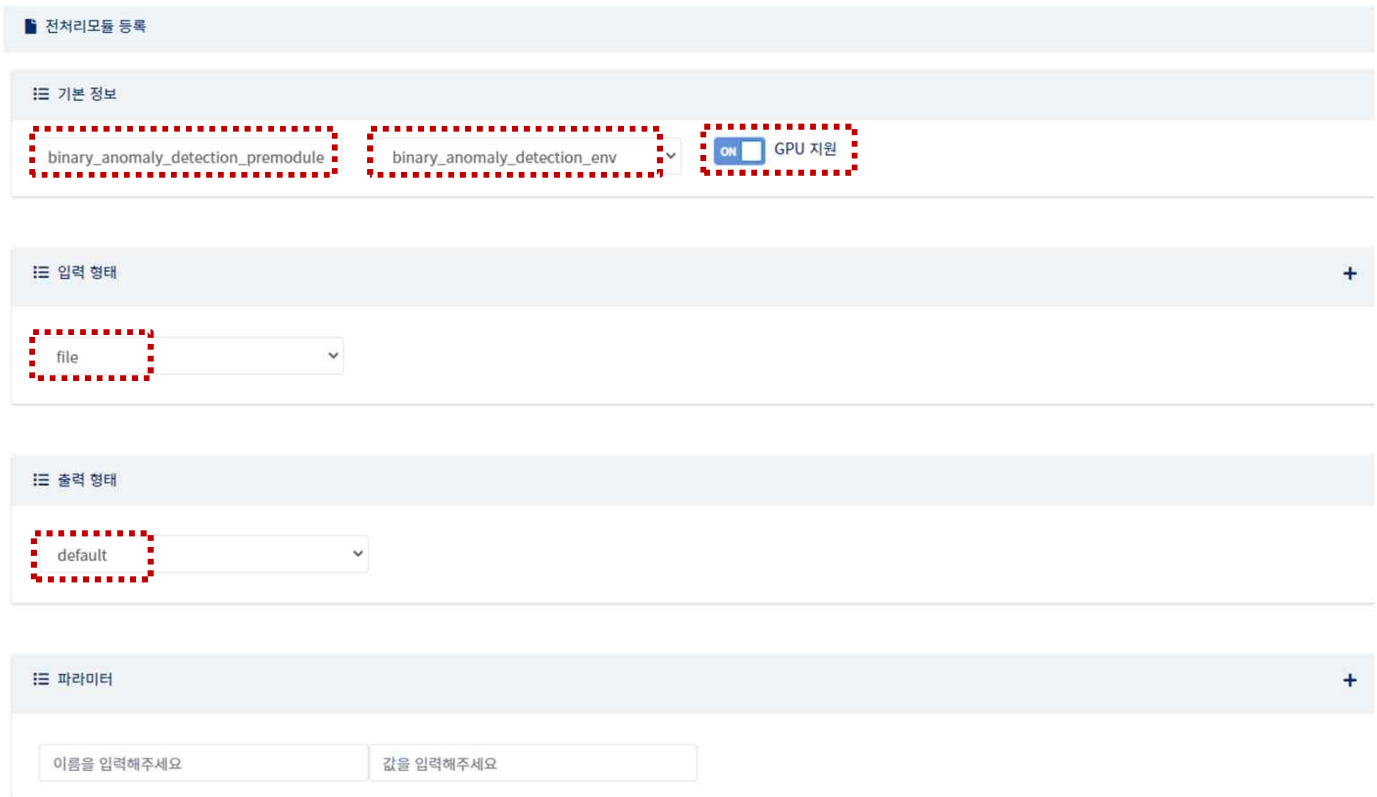
전처리명: binary_anomaly_detection_premodule

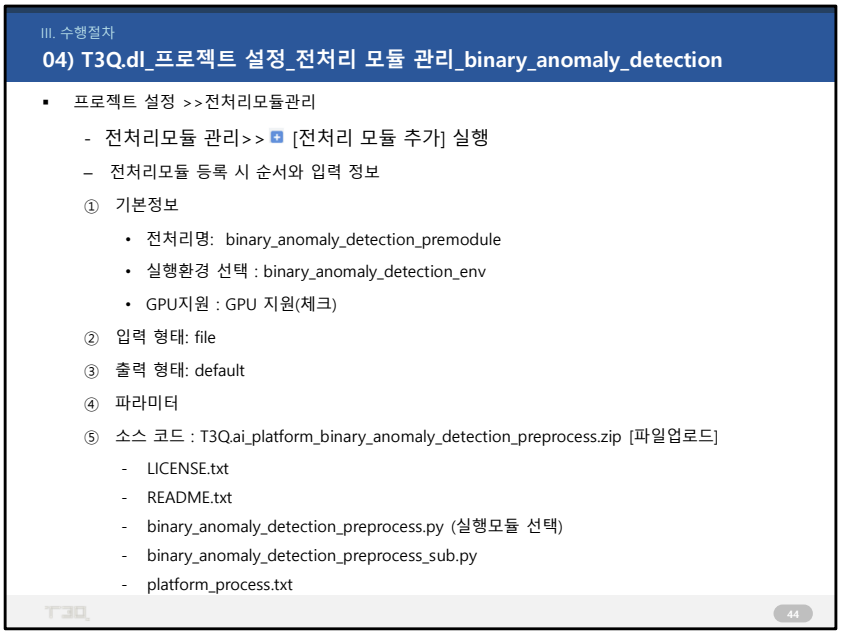
실행환경 선택 : binary_anomaly_detection_env

GPU지원 : GPU 지원(체크)

② 입력 형태: file

③ 출력 형태: default

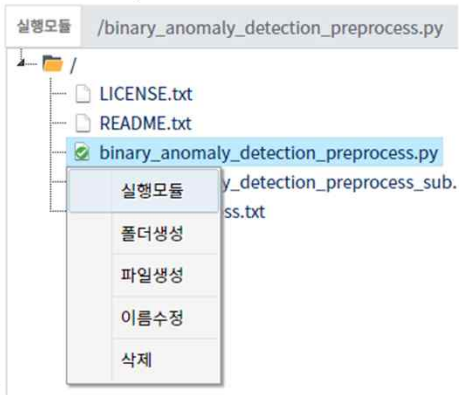




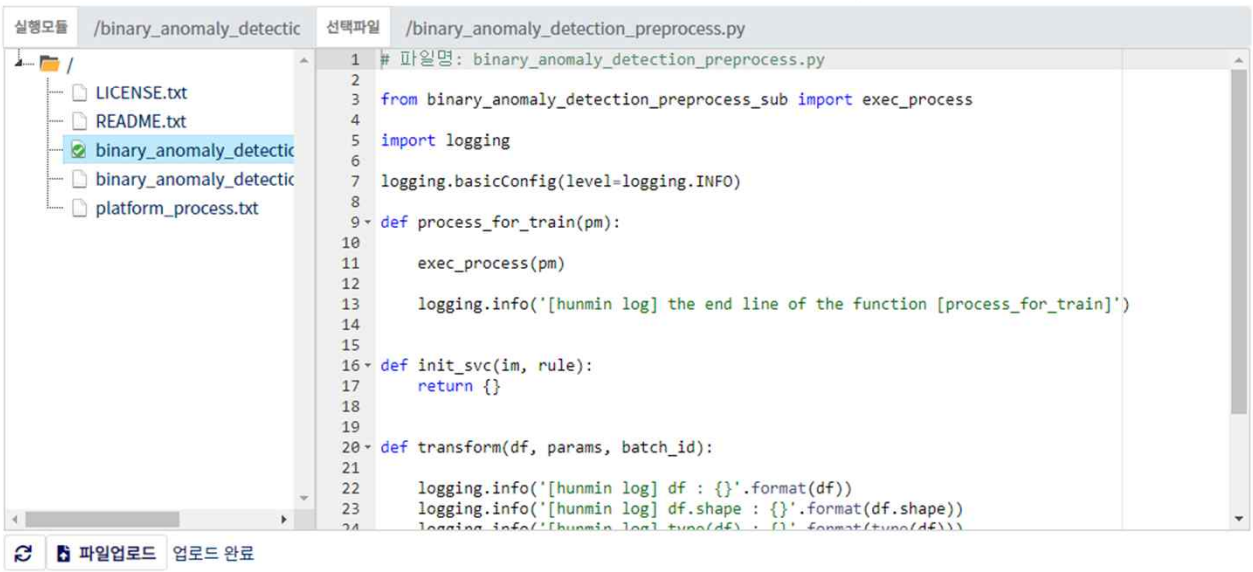
전처리모듈 등록 시 순서와 입력 정보

⑤ 소스 코드 : T3Q.ai_platform_binary_anomaly_detection_preprocess.zip [파일업로드] 누름

- LICENSE.txt
- README.txt
- binary_anomaly_detection_preprocess.py (실행모듈 선택)
- binary_anomaly_detection_preprocess_sub.py
- platform_process.txt



소스 코드



05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_binary_anomaly_detection

- 프로젝트 설정 >> 학습 알고리즘 관리
  - 학습 알고리즘 관리>>
    - [알고리즘 추가] 실행

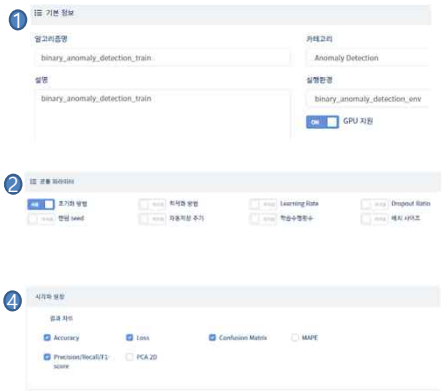
- 학습 알고리즘 등록 시 순서와 입력 정보

① 기본정보: 다음과 같이 입력
-알고리즘명:
binary_anomaly_detection_train
-설명 :
binary_anomaly_detection_train
-카테고리 : Anomaly Detection
-실행환경 :
binary_anomaly_detection_env
-GPU 지원 : 체크

② 공통 파라미터 : 초기화방법 체크

③ 모델 파라미터

④ 시각화 설정 : 아래 4개 항목만 체크
- Accuracy : 체크
- Loss : 체크
- Confusion Matrix : 체크
- Precision/Recall/F1-score : 체크

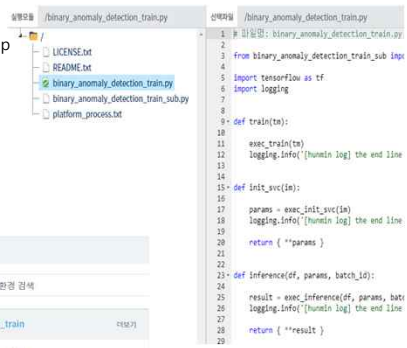


05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_binary_anomaly_detection

- 프로젝트 설정 >> 학습 알고리즘 관리
  - 학습 알고리즘 관리>> [알고리즘 추가] 실행
  - 학습 알고리즘 등록 시 순서와 입력 정보

⑤ 소스코드 업로드 : [파일업로드]

: T3Q.ai_platform_binary_anomaly_detection_train.zip
  - LICENSE.txt
  - README.txt
  - binary_anomaly_detection_train.py
  - binary_anomaly_detection_train_sub.py
  - platform_process.txt



: 실행 모듈 설정 >> [저장]
  - /binary_anomaly_detection_train.py
    실행 모듈 지정
  - [저장]을 누른다.

III. 수행절차

06) T3Q.dl_학습플랫폼_데이터셋 관리_binary_anomaly_detection

- 학습플랫폼 >> 데이터셋 관리
- 데이터셋 등록 절차
  - 데이터셋 관리>> 등록 실행
  - 데이터셋 명 : binary_anomaly_detection_dataset
  - 데이터셋 파일 : dataset.zip

III. 수행절차

07) T3Q.dl_학습플랫폼_전처리 모델 설계_binary_anomaly_detection

- 학습플랫폼 >> 전처리 모델 설계
- 전처리 모델 설계 절차

① 학습플랫폼 >> 데이터셋 관리
   - 데이터셋 명 : binary_anomaly_detection_dataset 선택

② binary_anomaly_detection_dataset 아래의 [전처리 모델 설계] 선택

07) T3Q.dl_학습플랫폼_전처리 모델 설계_binary_anomaly_detection

- 학습플랫폼 >> 전처리 모델 설계

② 전처리 모델 설계 절차

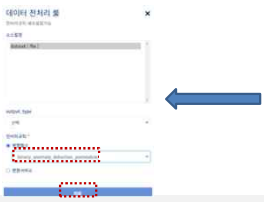
Step1. 기본 정보
- 모델명:binary_anomaly_detection_premodel

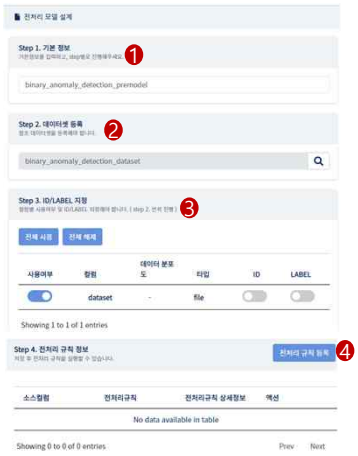
Step2. 데이터셋 등록:
- 참조데이터셋 :
  binary_anomaly_detection_dataset

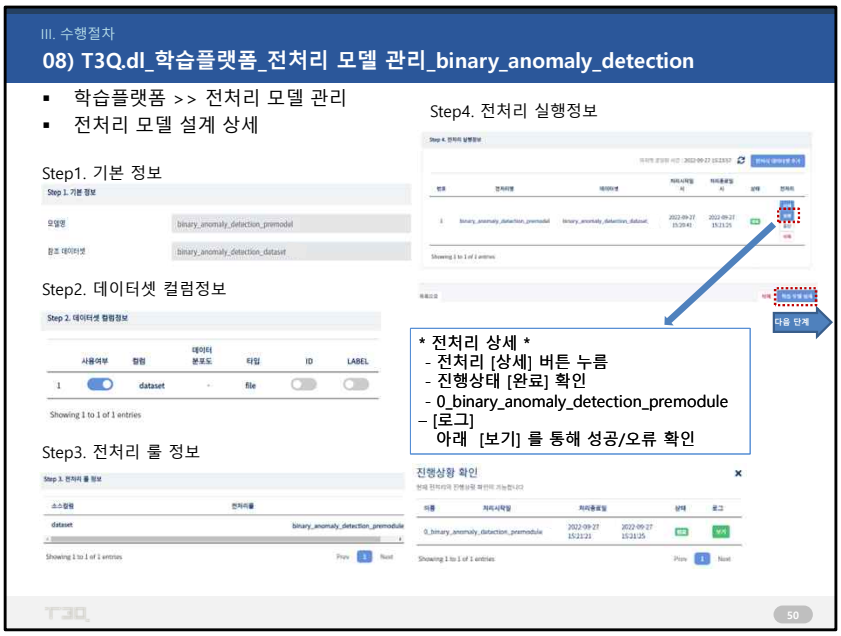
Step3. ID/LABEL 지정

Step4. 전처리 규칙 정보
- [전처리 규칙 등록] 선택

③ 전처리 규칙 등록 시 절차
- 소스컬럼 : dataset(file) 선택
- 변환 함수:binary_anomaly_detection_premodule 선택 후 [저장]





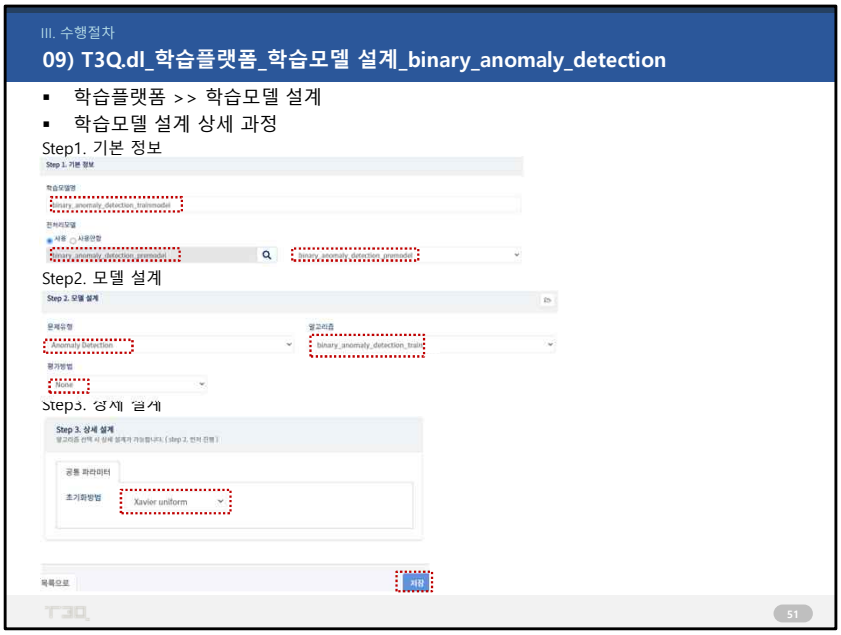


▪ 학습플랫폼 >> 전처리 모델 관리
▪ 전처리 모델 설계 상세
Step1. 기본 정보
- 모델명: binary_anomaly_detection_premodel
- 참조데이터셋 : binary_anomaly_detection_dataset
Step2. 데이터셋 컬럼정보
Step3. 전처리 룰 정보
Step4. 전처리 실행정보
전처리 상세
- 전처리 상세 버튼 누름
- 진행상황 확인
- 0_binary_anomaly_detection_premodule – [로그] 아래 [보기] 누름
[학습 모델 설계] 선택하여 다음 단계 진행

로그 확인

마지막 로딩된 시간 : 2022-09-27 16:16:49

```
2022-09-27 06:21:21,329 [ INFO] root: ### preprocessing start ###
2022-09-27 06:21:21,331 [ INFO] root: params={'pre_dataset_id': 1043, 'rule': {'source_column': ['dataset'], 'rule': 'preModel', 'rule_type':
'binary_anomaly_detection_premodule_v1', 'mod': 'U', 'param': {}, 'rule_no': '0', 'source_type': ['file'], 'module_info': '{"deploy_dt": "2022-
08-29 19:18:54", "template": "Python", "version": "1.0", "status": "deployed", "image_name": 362, "module_name":
"binary_anomaly_detection_preprocess"}', 'output_type': ['default']}, 'do_fit': True, 'test_no': None, 'test_dataset_path': None, 'log_path':
'/data/aip/logs'}
2022-09-27 06:21:21,368 [ WARN] root: datasource_repo_id : 159, datasource_repo_obj : <DataSourceRepo 159>, repo_type : path
2022-09-27 06:21:21,388 [ INFO] root: module_path=/data/aip/logs/t3qai/premodule/premodule_731/1
2022-09-27 06:21:21,463 [ INFO] root: dp_module=<module 'binary_anomaly_detection_preprocess' from
'/data/aip/logs/t3qai/premodule/premodule_731/1/binary_anomaly_detection_preprocess.py'>
2022-09-27 06:21:21,463 [ INFO] root: [hunmin log]  the start line of the function [exec_process]
2022-09-27 06:21:21,477 [ INFO] root: [hunmin log] Files and directories in
/data/aip/datalake/t3qai/AI_HUNMIN/binary_anomaly_detection/collection :
2022-09-27 06:21:21,477 [ INFO] root: [hunmin log] dir_list : ['dataset.zip']
2022-09-27 06:21:25,856 [ INFO] root: [hunmin log] Files and directories in /data/aip/dataset/t3qai/pm/pm_1034/ds_1043 :
2022-09-27 06:21:25,856 [ INFO] root: [hunmin log] dir_list : ['dataset']
```



학습플랫폼 >> 학습모델 설계

1. AI 훈민정음 >> 학습플랫폼 >> 학습모델 설계 상세 과정
   1) Step 1. 기본 정보
      학습모델명 binary_anomaly_detection_trainmodel
      전처리모델
      [사용] 체크
      binary_anomaly_detection_premodel binary_anomaly_detection_premodel

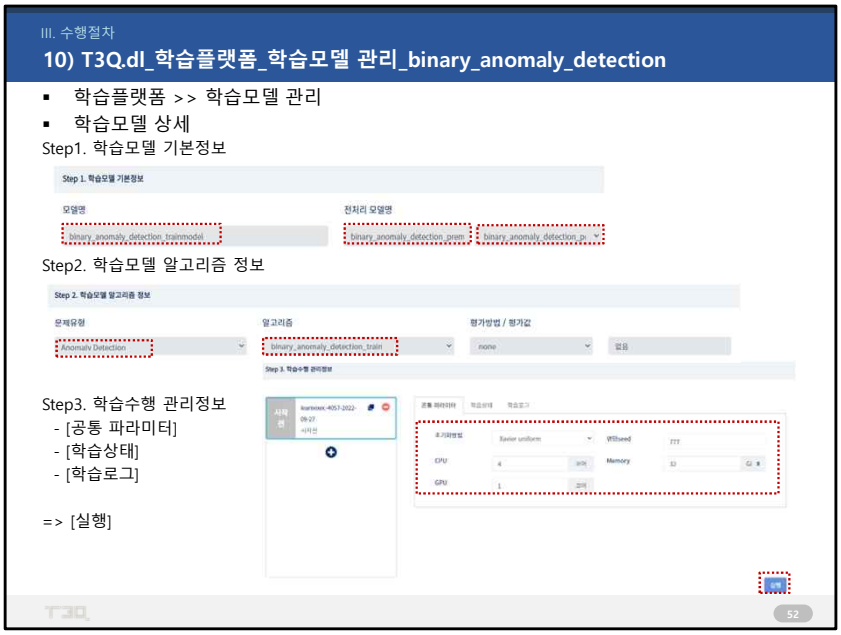
   2) Step 2. 모델 설계
      문제유형 Anomaly Detection
      알고리즘 binary_anomaly_detection_train
      평가방법
      Train-Test Split : None

   3) Step 3. 상세 설계
      알고리즘 선택 시 상세 설계가 가능합니다. (step 2. 먼저 진행)

      (1) 공통 파라미터
      초기화방법 : Xavier uniform

   4) [저장] 누름

2. AI 훈민정음 >> 학습플랫폼 >> [학습모델 관리] 에서 등록된 학습 모델 확인



학습플랫폼 >> 학습모델 관리

1. 학습플랫폼 >> 학습모델 관리

2 학습모델 상세

1) Step 1. 학습모델 기본정보
모델명 binary_anomaly_detection_trainmodel
전처리 모델명 binary_anomaly_detection_premodel
binary_anomaly_detection_premodel

2) Step 2. 학습모델 알고리즘 정보
문제유형 Anomaly Detection
알고리즘 binary_anomaly_detection_train
평가방법 / 평가값 None

3) Step 3. 학습수행 관리정보
(1) 공통 파라미터

초기화방법	Xavier uniform
랜덤seed	777
CPU	4 코어
Memory	32 Gi
GPU	1 코어

(2) 학습상태
학습상태 시작전 [- ~ -]
Accuracy Loss Confusion Matrix Precision/Recall/F1-score

[실행] 버튼 누름

III. 수행절차

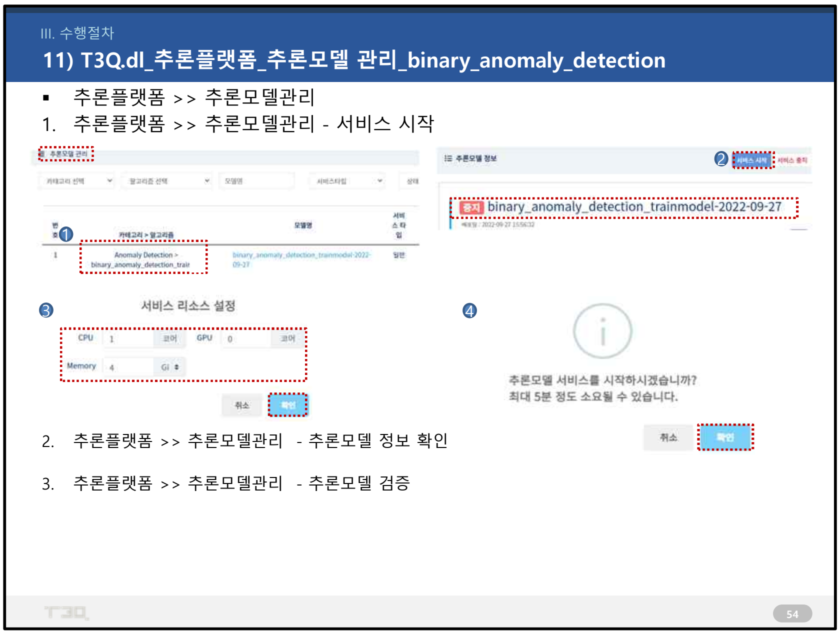
10) T3Q.dl_프로젝트 설정_학습모델 관리_binary_anomaly_detection

- 학습플랫폼 >> 학습모델 관리
- 학습모델 상세
- [실행] 완료 후

- [학습상태]

- [학습로그]

- 모델 배포 : [모델 배포] 버튼 누르고 [배포] 누름

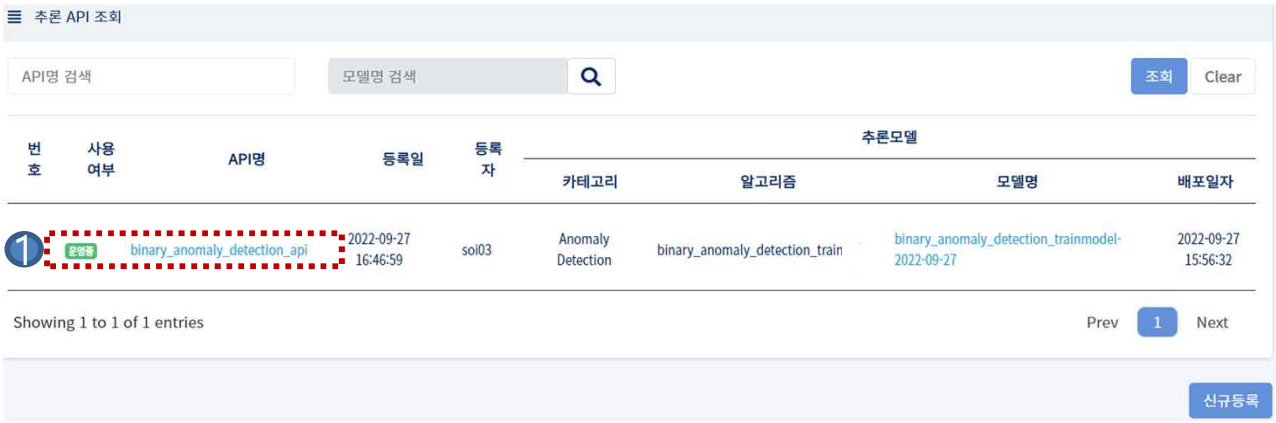


추론플랫폼 >> 추론모델관리

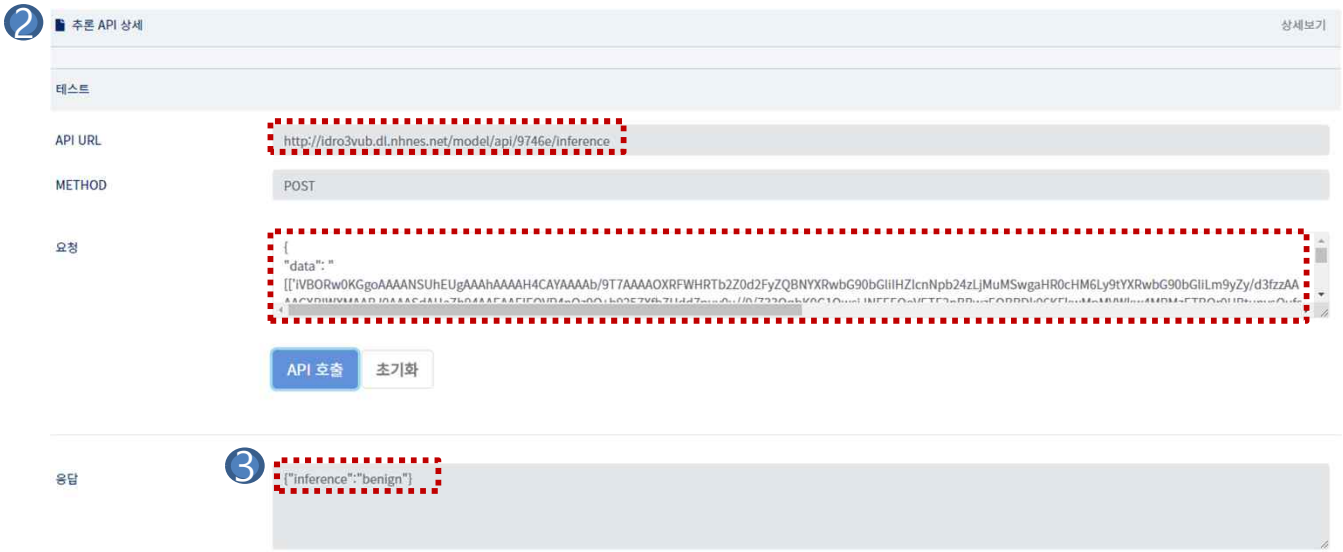
1. 추론플랫폼 >> 추론모델관리 - 서비스 시작
2. 추론플랫폼 >> 추론모델관리 – 추론모델 정보 확인

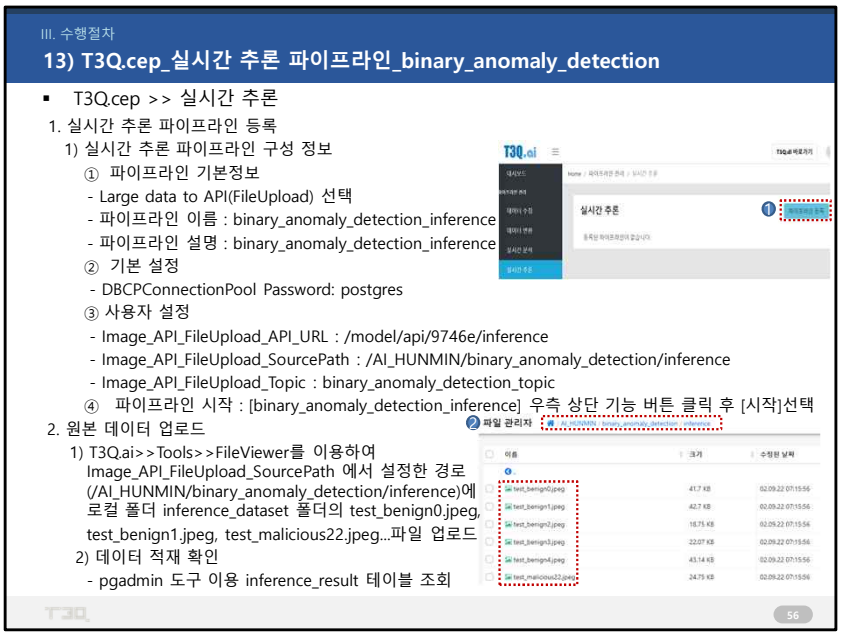
3. 추론플랫폼 >> 추론모델관리 - 추론모델 검증
 [추론모델테스트]-[요청] 에 아래의 값을 넣고, [테스트] 눌러 수행
요청 : 입력 예시 : [['/data/aip/file_group/pm/pm_334/ds_441/image/1/1230.png']]
요청 : '(11_1) Request_binary_anomaly_detection.txt' 파일 안에 있는 base64 형식의 데이터 입력
응답 : "{₩"inference₩":₩"benign₩"}₩n"

▪추론플랫폼 >> 추론API관리
1.추론플랫폼 >> 추론API관리 – 신규 등록
2.추론플랫폼 >> 추론API관리 – 추론 API 상세



=> 요청 :{"data":" [['테스트 데이터 값 입력='']]"}=>[API 호출] 클릭
=> 응답 : {"data":["결과"]}





(2) 데이터 적재 확인
T3Q.ai >> Tools>> PgAdmin

#inference_origin
inference_result

테이블 조회

#select * from inference_origin;

select * from inference_result;

데이터 저장 확인

==========
SELECT * FROM public.inference_result
where url like '%/model/api/9746e/inference%'
order by start_time desc