Anomaly_Detection으로 Crack Detection 수행하기

Alibi Detect를 이용하여 콘크리트 바닥의 Crack을 검출해내는 것을 목표로 한다.

Dataset은 Kaggle의 Surface-Crack-Dataset을 이용하였다.

https://www.kaggle.com/arunrk7/surface-crack-detection

Negative Image와 Positive Image 2종류가 있는데, 여기서 Negative는 정상적인 Surface, Normal Image를 의미한다.

반대로 Positive Image가 Cracked, Crack Image이다.

실제 현업에서는 정상적인 데이터에 비해 비정상데이터의 충분한 데이터를 확보하기 어렵다.

이를 보완하기위해 '정상적인 데이터만 학습' 한 후, 정상 데이터와 input image의 픽셀값의 차이를 이용하여 큰 차이가 나는 부분을 표현하는 방법을 사용한다.

여기에 Auto-Encoder를 통해 이미지를 재구축하여 사용한다.

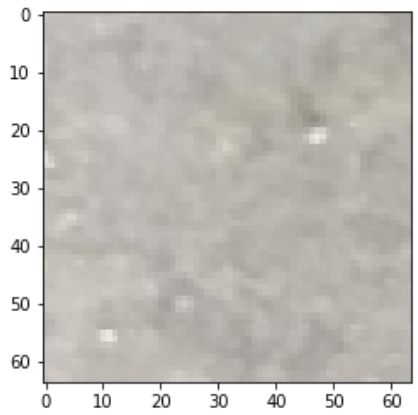
이러한 방식으로 다양한 데이터셋을 사용하여 구현할수있다.

```
## Alibi-Detect Module Install ##
!pip install alibi_detect

## Image Pre-Processing ##
import numpy as np
from glob import glob
from sklearn.model_selection import train_test_split
from PIL import Image

img_list = glob('C:/Users/user/Desktop/data/Surface_Crack_Dataset/Negative/*.jpg') # normal 이미지

train_img_list, val_img_list = train_test_split(img_list, test_size=0.1, random_state=2021)
print(len(train_img_list), len(val_img_list))

def img_to_np(fpaths, resize=True):
  img_array = []
  for fname in fpaths:
    try:
      img = Image.open(fname).convert('RGB')
      if resize:
        img = img.resize((64,64)) # 64x64 size 변환!
        img_array.append(np.asarray(img)) # numpy 배열 변환
    except:
      continue  # try 문에서 해당되지 않으면 넘김!

  images = np.array(img_array)
  return images
------------------------------------------------------------
18000 2000
```

```
# train 1000장, val 32장 Normalize
x_train = img_to_np(train_img_list[:1000])
x_train = x_train.astype(np.float32) / 255. # normalize

x_val = img_to_np(val_img_list[:32])
x_val = x_val.astype(np.float32) / 255.

print(x_train.shape)
print(x_val.shape)
```

```
-----------------------------------------------------
(1000, 64, 64, 3)
(32, 64, 64, 3)
```

```
import matplotlib.pyplot as plt

plt.imshow(x_train[0])
```




```
## Reconstruction 이미지를 생성하는 Auto-Encoder 학습 ##
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, Dense, Layer, Reshape, InputLayer
from alibi_detect.od import OutlierVAE
latent_dim = 1024

encoder_net = tf.keras.Sequential([
                                    InputLayer(input_shape=(64,64,3)),
                                    Conv2D(64, 4, strides=2, padding='same', activation=tf.nn.relu),
                                    Conv2D(128, 4, strides=2, padding='same', activation=tf.nn.relu),
                                    Conv2D(512, 4, strides=2, padding='same', activation=tf.nn.relu)
])

# decoder의 input은 latent vector 크기로 들어간다는 점!
# covloution transpose 과정은 차원을 늘리는 과정
# deconv 과정을 거치면 결국 input size와 같은 64x64x3 의 size로 나옴.
decoder_net = tf.keras.Sequential([
                                    InputLayer(input_shape=(latent_dim, )),
                                    Dense(4 * 4 * 128),
                                    Reshape(target_shape=(4, 4, 128)), # convolution transepose 하기 위해
                                    Conv2DTranspose(256, 4, strides=2, padding='same', activation=tf.nn.relu),
                                    Conv2DTranspose(64, 4, strides=2, padding='same', activation=tf.nn.relu),
```

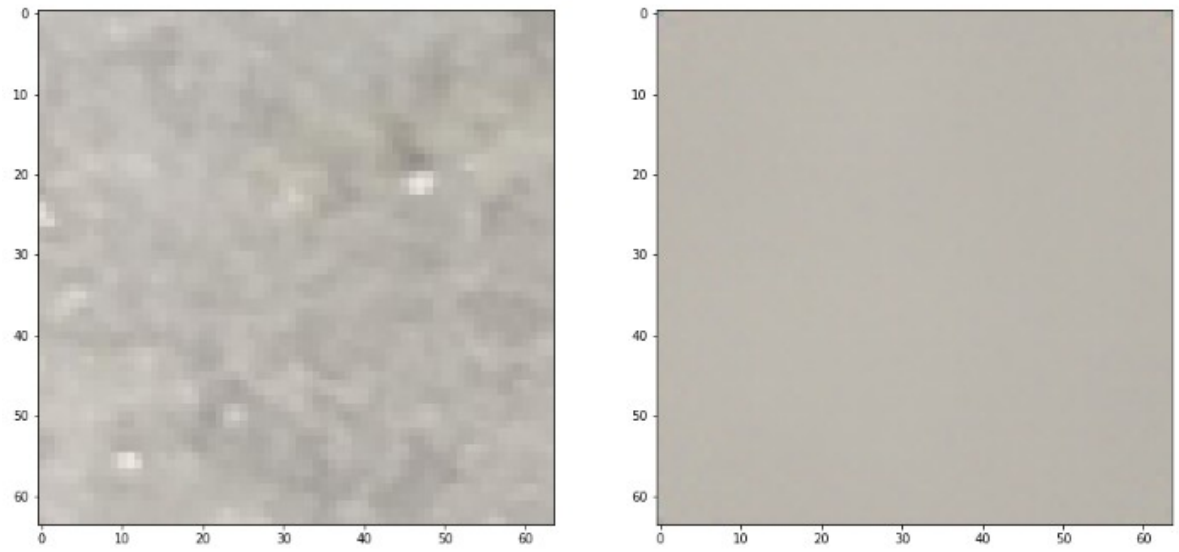
```
                                    Conv2DTranspose(32, 4, strides=2, padding='same', activation=tf.nn.relu),
                                    Conv2DTranspose(3, 4, strides=2, padding='same', activation='sigmoid')
])

outlier = OutlierVAE(
    threshold=.005, # instance(픽셀의 차) score
    score_type='mse',
    encoder_net=encoder_net,
    decoder_net=decoder_net,
    latent_dim=latent_dim
)
```

```
# 정상데이터만을 넣어서 Recontruction 이미지를 만들어냄
outlier.fit(
    x_train,
    epochs=30,
    verbose=True  # '학습 결과를 출력해라'
)
```

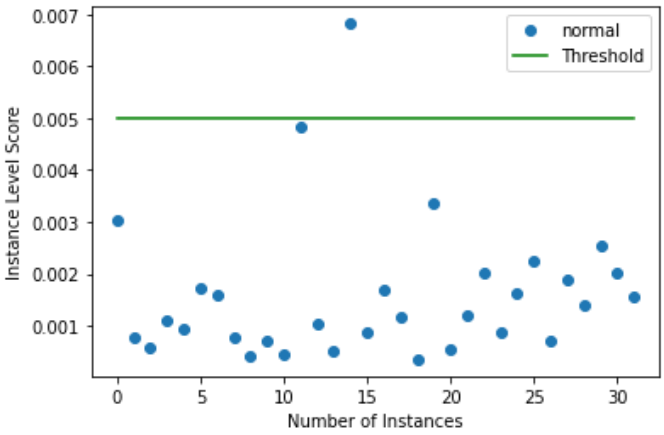
```
16/16 [=] - 11s 669ms/step - loss: 66499.9216
16/16 [=] - 11s 679ms/step - loss: 5540.4471
16/16 [=] - 11s 682ms/step - loss: -2378.5647
16/16 [=] - 11s 692ms/step - loss: -8955.6741
16/16 [=] - 11s 668ms/step - loss: -14111.0999
16/16 [=] - 11s 663ms/step - loss: -14383.1048
16/16 [=] - 11s 662ms/step - loss: -15386.8731
16/16 [=] - 11s 671ms/step - loss: -19588.4047
16/16 [=] - 11s 665ms/step - loss: -20344.0396
16/16 [=] - 11s 676ms/step - loss: -20982.5631
16/16 [=] - 11s 669ms/step - loss: -20746.1324
16/16 [=] - 11s 667ms/step - loss: -20972.0537
16/16 [=] - 11s 671ms/step - loss: -20885.3540
16/16 [=] - 11s 665ms/step - loss: -20987.7122
16/16 [=] - 11s 664ms/step - loss: -21080.7340
16/16 [=] - 11s 685ms/step - loss: -21253.9436
16/16 [=] - 10s 650ms/step - loss: -21178.5016
16/16 [=] - 10s 651ms/step - loss: -21199.7065
16/16 [=] - 11s 664ms/step - loss: -21604.3486
16/16 [=] - 10s 646ms/step - loss: -21608.0299
16/16 [=] - 10s 637ms/step - loss: -21455.6215
16/16 [=] - 10s 640ms/step - loss: -21500.3040
16/16 [=] - 10s 643ms/step - loss: -21480.2590
16/16 [=] - 10s 641ms/step - loss: -21731.7845
16/16 [=] - 10s 639ms/step - loss: -21777.1136
16/16 [=] - 10s 642ms/step - loss: -21606.8129
16/16 [=] - 10s 636ms/step - loss: -21667.9277
16/16 [=] - 10s 639ms/step - loss: -21635.0533
16/16 [=] - 10s 651ms/step - loss: -21619.5635
16/16 [=] - 10s 652ms/step - loss: -21498.1449
```

```
# 원본 train 이미지
x = x_train[0].reshape(1, 64, 64, 3)

# Reconstrution 이미지
x_reconst = outlier.vae(x).numpy()
```

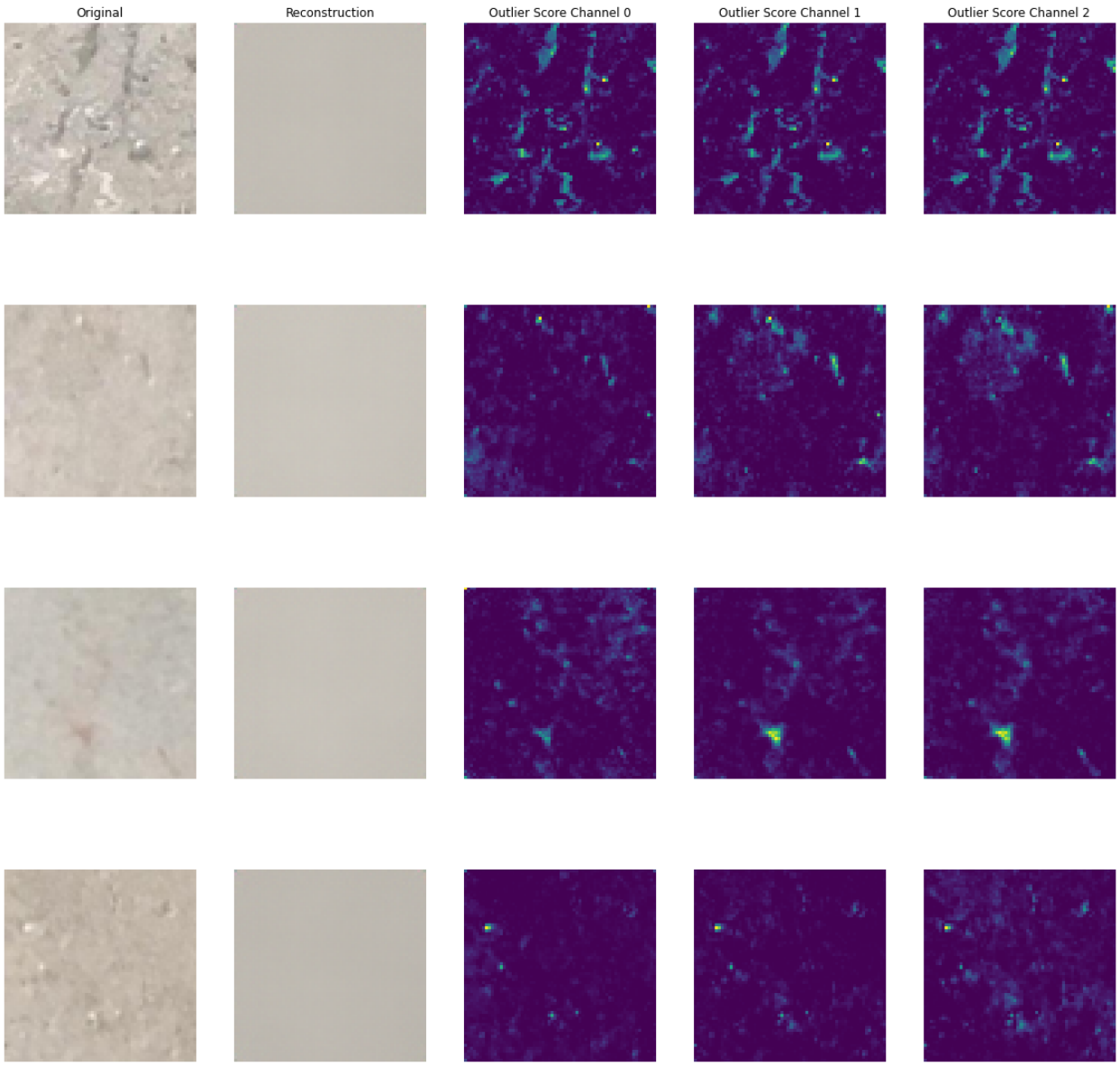
```
# imshow
fig, axes = plt.subplots(1, 2, figsize=(16,10)) # 1행 2열의 도표 생성!
axes[0].imshow(x.squeeze())
axes[1].imshow(x_reconst.squeeze())
```



```
## Normal 데이터 Test 과정(Validation) ##
# 정상데이터로만 학습하여 생성한 인코더가 재생성 이미지를 잘만들어 내는지를 확인하는 과정
# Alibi Detect 모듈을 통해 그래프를 그릴수 있음
# 그래프의 파란색 점은 각 validation score를 표시한 것 (현재는 32개의 점)
# 그래서 threshold 보다 아래쪽에 있으면 normal 데이터임을 나타냄
# 픽셀간의 차를 나타낸 instance score

from alibi_detect.utils.visualize import plot_instance_score, plot_feature_outlier_image
# 정상데이터로만 생성한 encoder가 재생성 이미지를 잘만들어내는지?
outlier_preds = outlier.predict(
    x_val,
    outlier_type='instance', # 픽셀 간의 차(instance)
    return_feature_score=True,
    return_instance_score=True
)

target = np.zeros(x_val.shape[0],).astype(int)
labels = ['normal', 'outlier']
plot_instance_score(outlier_preds, target, labels, outlier.threshold)
```



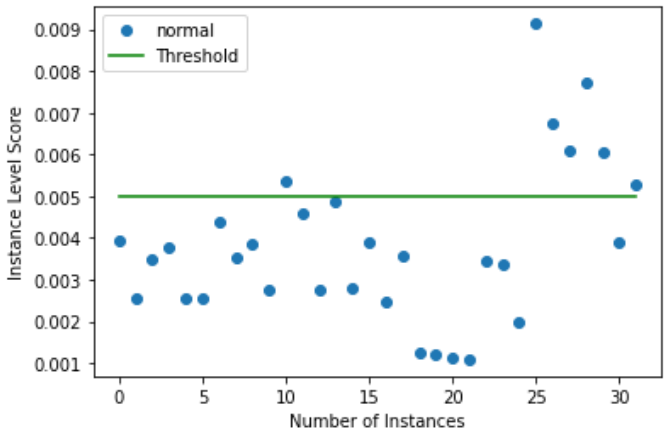

```
# outlier score channel 표시된 부분은 Original과 Reconstructions 부분의 차를 뺀 것

x_recon = outlier.vae(x_val).numpy()

plot_feature_outlier_image(
    outlier_preds,
    x_val,
    X_recon=x_recon,
    max_instances=4,
    outliers_only=False  # outliers_only=True는 위에서 그린 그래프에서 Threshold 벗어난 그림
)
```



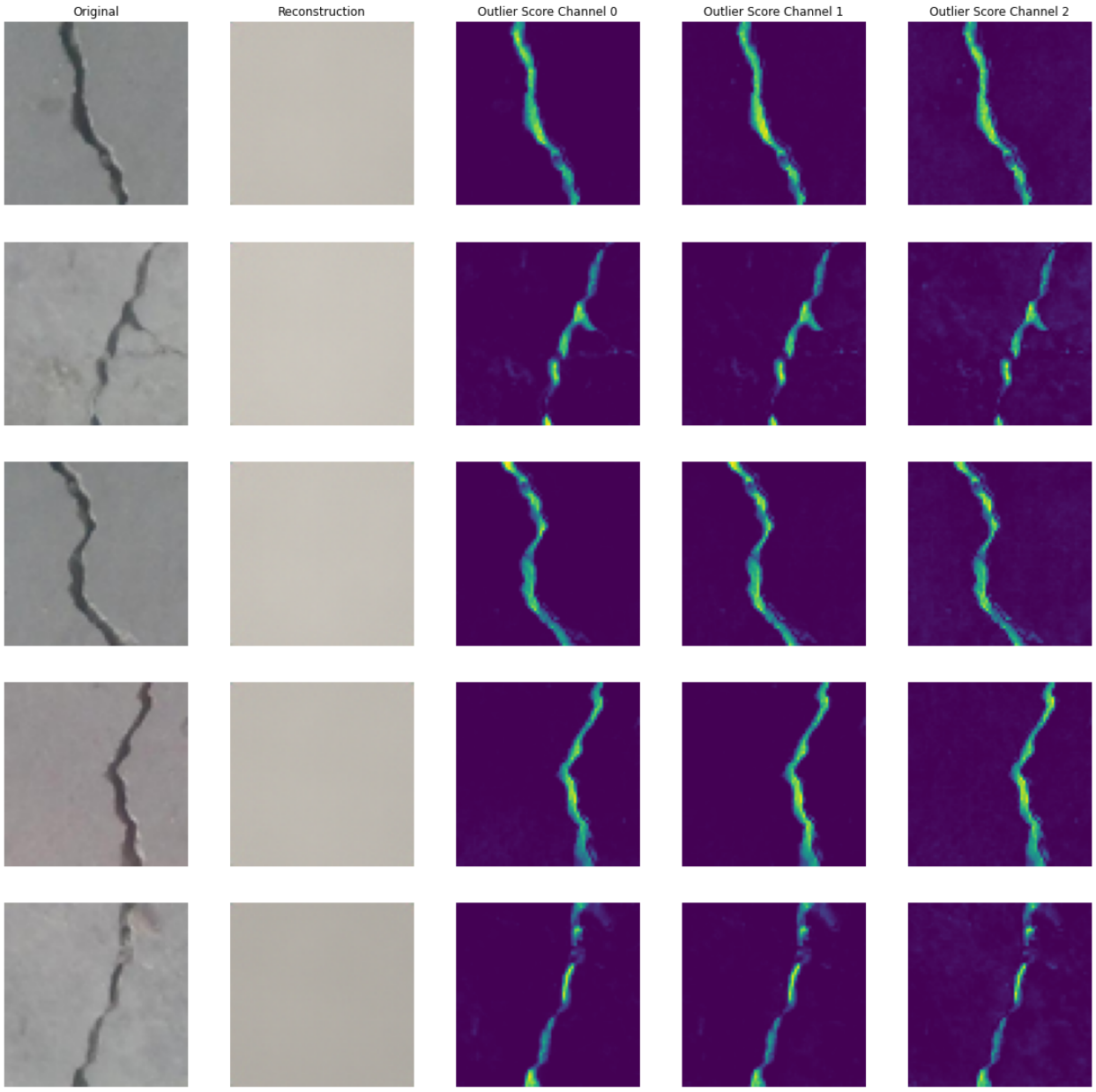

```
## Crack 데이터 Test 과정 ##
# Crack 데이터와 재생성한 이미지의 차이를 구함 #
# Crack의 Feature를 얻는 과정 #
test_img_list = glob('C:/Users/user/Desktop/data/Surface_Crack_Dataset/Positive/*.jpg')

x_test = img_to_np(test_img_list[:32])
x_test = x_test.astype(np.float32) / 255.

print(x_test.shape)
-------------------------------------------------
(32, 64, 64, 3)
```

```
outlier_preds = outlier.predict(
    x_test,
```

```
    outlier_type='instance',
    return_feature_score=True,
    return_instance_score=True
)

target = np.zeros(x_test.shape[0],).astype(int)
labels = ['normal', 'outlier']
plot_instance_score(outlier_preds, target, labels, outlier.threshold)
```




```
# instance score가 threshold 를 벗어난 그림들을 표현한 것!
x_reconst = outlier.vae(x_test).numpy()

plot_feature_outlier_image(
    outlier_preds,
    x_test,
    X_recon=x_recon,
    max_instances=5,
    outliers_only=False
)
```


Original
Reconstruction
Outlier Score Channel 0
Outlier Score Channel 1
Outlier Score Channel 2