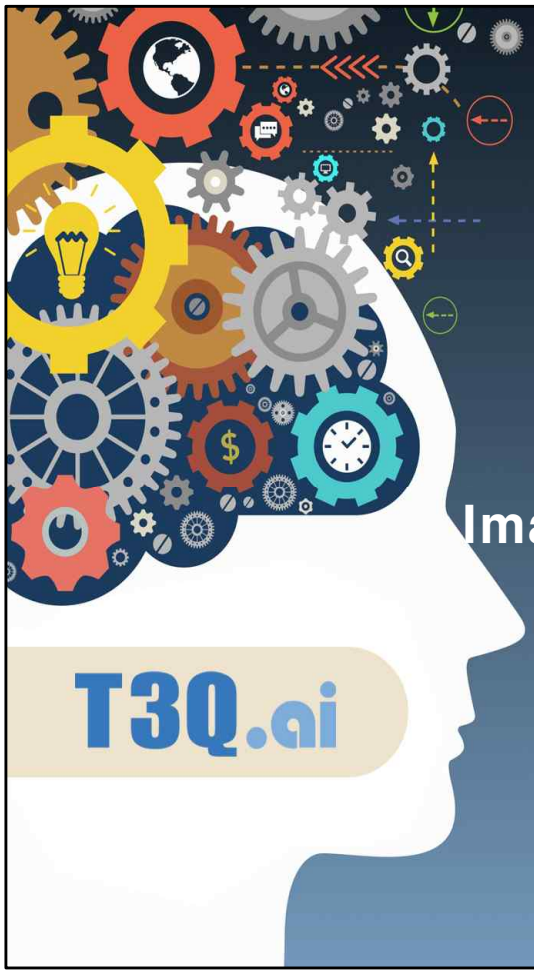


III장. AI훈민정음_5
Image_Generation (Content Image를 Style Image의 화풍으로 이미지 생성)

목 차

2-2 image generation

김홍도 화가의 작품인 금강사군첩을 Neural style transfer를 이용하여
입력된 이미지로부터 새로운 이미지를 생성하는 예제

1. 데이터셋

Content Image :
카페안에서 밖을 촬영한 사진으로, 하늘과 나무 등 카페 밖 풍경이 나타난 사진

Style Image : 금강사군첩
60폭으로 이루어진 화첩으로 금강산 및 관동 8경 지역을 그린 김홍도 화가의 작품

2. 전처리 및 학습

전처리:
이미지 크기 조정 및 Antialias 적용
Content image :(1440,1080) → (720,540)
Style image : (850,590) → (425,295)

학습:
사전에 훈련된 CNN(VGG19)모델을 활용한 Neural Style Transfer 모델

3. 추론 결과

Content Image에서 형태를 추출하고
Style Image에서 추출된 특징을 적용하여
새로운 이미지 생성

4. 기대 효과

동양화 뿐만 아니라 다른 유명 화가의 작품에도 적용 가능

데이터셋

- Content Image : 이미지의 content 정보가 담긴 형태만 추출
- Style Image : 이미지의 style 정보가 담긴 특징만 추출

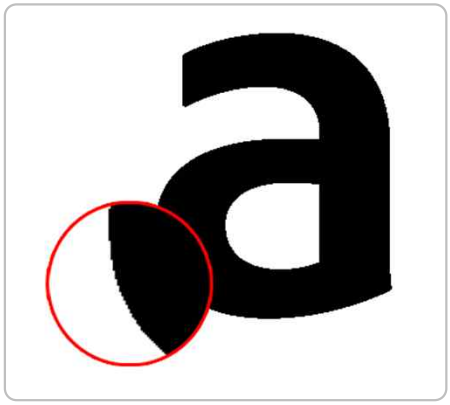
Content Image

Style Image

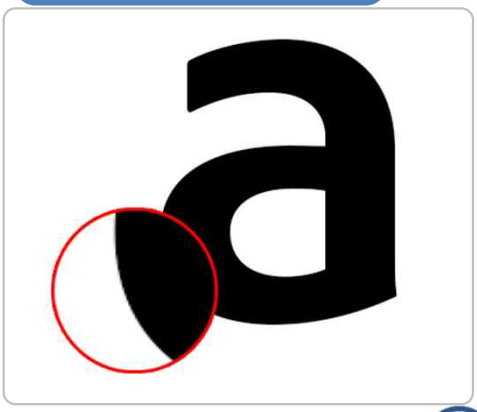
전처리 - Antialias

- Antialias : 높은 해상도에서 낮은 해상도로 이미지를 변환할 때 픽셀이 깨지는 현상을 최소화해주는 기법

Resize한 이미지

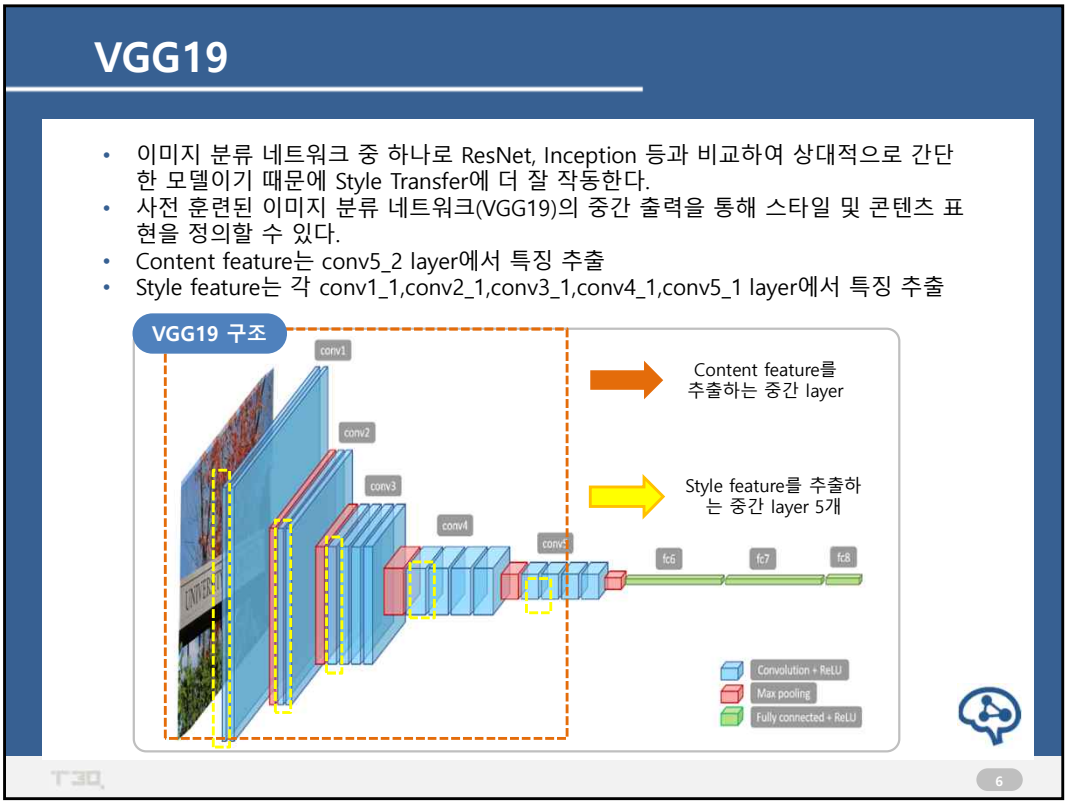


Resize 후 Antialias적용한 이미지



5

이미지 출처 : https://helpx.adobe.com/photoshop-elements/key-concepts/aliasing-anti-aliasing.html(Adobe공식 홈페이지)



이미지 출처: https://colab.research.google.com/github/mamaj/cnn-featurevis-ece421/blob/master/CNNFeatureVis_handout.ipynb

Convolution/Pooling(이미지 객체가 라벨에 속할 확률 계산)

0) 인풋: 224 x 224 x 3 이미지(224 x 224 RGB 이미지) 입력 받음

1) 1층(conv1_1): 64개의 3 x 3 x 3 필터 커널 사용. zero padding은 1만큼 해줬고, 간격(stride)은 1로 설정함. 64장의 224 x 224 특성 맵(224 x 224 x 64)들이 생성됨

2) 2층(conv1_2): 64개의 3 x 3 x 64 필터 커널 사용. 64장의 224 x 224 특성 맵들(224 x 224 x 64)이 생성됨. 그 후 2 x 2 최대 pooling을 2간격으로 적용해 특성 맵의 사이즈를 112 x 112 x 64로 줄임

3) 3층(conv2_1): 128개의 3 x 3 x 64 필터 커널 사용. 128장의 112 x 112 특성 맵들(112 x 112 x 128)이 생성됨.

4) 4층(conv2_2): 128개의 3 x 3 x 128 필터 커널로 사용. 128장의 112 x 112 특성 맵들(112 x 112 x 128)이 생성됨. 그 후 pooling을 통해 특성 맵 사이즈를 56 x 56 x 128로 줄임.

5) 5층(conv3_1): 256개의 3 x 3 x 128 필터 커널 사용. 256장의 56 x 56 특성 맵들(56 x 56 x 256) 생성됨

6) 6층(conv3_2): 256개의 3 x 3 x 256 필터 커널 사용. 256장의 56 x 56 특성 맵들(56 x 56 x 256) 생성됨

7) 7층(conv3_3): 256개의 3 x 3 x 256 필터 커널 사용. 256장의 56 x 56 특성 맵들(56 x 56 x 256) 생성됨.

8) 8층(conv3_4): 256개의 3 x 3 x 256 필터 커널 사용. 256장의 56 x 56 특성 맵들(56 x 56 x 256) 생성됨. 그 후 pooling을 통해 특성 맵 사이즈를 28 x 28 x 256로 줄임

9) 9층(conv4_1): 512개의 3 x 3 x 256 필터 커널 사용. 512장의 28 x 28 특성 맵들(28 x 28 x 512)이 생성됨

10) 10층(conv4_2): 512개의 3 x 3 x 512 필터 커널 사용. 512장의 28 x 28 특성 맵들(28 x 28 x 512)이 생성됨

11) 11층(conv4_3): 512개의 3 x 3 x 512 필터 커널 사용. 512장의 28 x 28 특성 맵들(28 x 28 x 512)이 생성됨.

12) 12층(conv4_4): 512개의 3 x 3 x 512 필터 커널 사용. 512장의 28 x 28 특성 맵들(28 x 28 x 512)이 생성됨. 그 후 pooling을 통해 특성 맵 사이즈를 14 x 14 x 512로 줄임

13) 13층(conv5_1): 512개의 3 x 3 x 512 필터 커널 사용. 512장의 14 x 14 특성 맵들(14 x 14 x 512)이 생성됨

14) 14층(conv5_2): 512개의 3 x 3 x 512 필터 커널 사용. 512장의 14 x 14 특성 맵들(14 x 14 x 512)이 생성됨

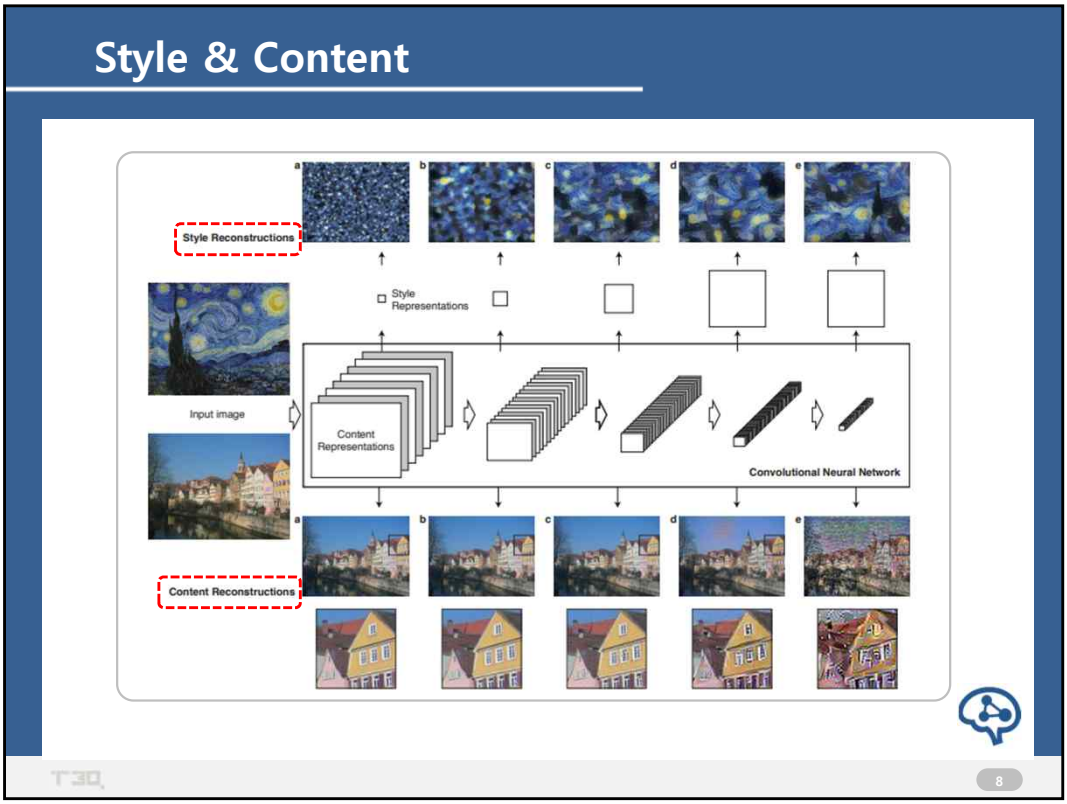
15) 15층(conv5-3): 512개의 3 x 3 x 512 필터 커널 사용. 512장의 14 x 14 특성 맵들(14 x 14 x 512)이 생성됨.

16) 16층(conv5-4): 512개의 3 x 3 x 512 필터 커널 사용. 512장의 14 x 14 특성 맵들(14 x 14 x 512)이 생성됨. 그 후 pooling을 통해 특성 맵 사이즈를 7 x 7 x 512로 줄임

Fully Connected Layer(이미지 분류하는데 사용되는 계층)
17) 17층(fc6): 7 x 7 x 512의 특성 맵 flatten 해줌. 7 x 7 x 512 = 25088개의 뉴런이 되고, fc6층의 4096개의 뉴런과 fully connected 됨. 훈련 시 dropout이 적용됨.

18) 18층(fc7): 4096개의 뉴런으로 구성되어 fc6층의 4096개의 뉴런과 fully connected 됨. 훈련 시 dropout이 적용됨.

19) 19층(fc8): 1000개의 뉴런으로 구성됨. fc7층의 4096개의 뉴런과 fully connected 됨. 1000개의 뉴런으로 구성되어 1000개의 클래스로 분류하는 것이 가능



이미지 출처 : Image Style Transfer Using Convolutional Neural Networks(CVPR 2016) 논문

Style Transfer는 content image와 style image를 cnn에 input으로 넣고 **Style Reconstructions** 과 **Content Reconstructions** 과정을 거쳐 새로운 이미지를 생성하는 모델이다.

Style Reconstructions

- vgg19 network에 style image를 input image로 넣어 각 layer마다 style 특징을 추출(a~e)하여 추출된 style image의 Gram Matrix 값과 재생성 되는 이미지의 Gram Matrix 값을 비교한다.
- Gram Matrix는 하나의 이미지로부터 얻은 각 layer에서 feature 간의 상관관계(하나의 style에 대한 정보)를 말하며, 이 값 자체를 이미지의 style이라고 볼 수 있다.
- Style Reconstructions은 두 이미지의 Gram Matrix가 유사해지도록 업데이트 하여 style 정보를 재구성하는 과정을 말한다.

- CNN layer('conv1 1' (a), 'conv1 1' and 'conv2 1' (b), 'conv1 1', 'conv2 1' and 'conv3 1' (c), 'conv1 1', 'conv2 1', 'conv3 1' and 'conv4 1' (d), 'conv1 1', 'conv2 1', 'conv3 1', 'conv4 1' and 'conv5 1' (e).) 의 다른 하위 집합에 구축된 스타일 표현에서 입력 이미지의 스타일을 재구성한다. 이렇게 하면 장면의 전체 배열에 대한 정보를 버리는 동시에 주어진 이미지의 스타일과 유사하도록 이미지가 생성된다.

Content Reconstructions

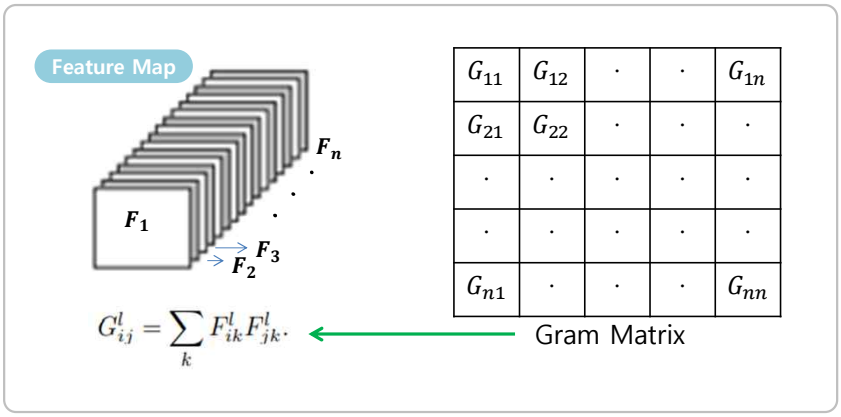
- vgg19 network에 content image를 넣어 layer층에서의 추출되는 특징(a~e) 값이 재생성 되는 이미지의 특징 값과 비슷해지도록 업데이트 하는 과정을 말한다.

- 네트워크의 특정 계층에서 출력 값으로 입력 이미지를 재구성한 정보를 a-e 그림과 같이 CNN의 여러 층에서 시각화할 수 있다.
- 원본의 'conv1 2'(a), 'conv2 2'(b), 'conv3 2'(c), 'conv4 2'(d) 및 'conv5 2'(e) layer에서 입력 이미지를 재구성한다. 이 과정을 통해 하위 layer(a–c)로부터의 재구성이 거의 완벽하다는 것을 발견했고, 네트워크의 상위 layer(d,e)에서는 이미지의 콘텐츠의

자세한 픽셀 정보가 손실된다는 것을 알 수 있다.

Gram Matrix

- 네트워크를 통과하는 이미지의 각 feature map간 상관관계를 나타내는 행렬
- Gram Matrix는 0-1 사이 값으로 표현되며, 이 값은 1에 가까울수록 상관관계가 크고 0에 가까울수록 상관관계가 작다는 것을 나타낸다.
- 상관관계가 클수록 특징간의 유사점이 많음을 의미한다.



$$\begin{bmatrix} G_{11} & G_{12} & \cdot & \cdot & G_{1n} \\ G_{21} & G_{22} & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ G_{n1} & \cdot & \cdot & \cdot & G_{nn} \end{bmatrix}$$

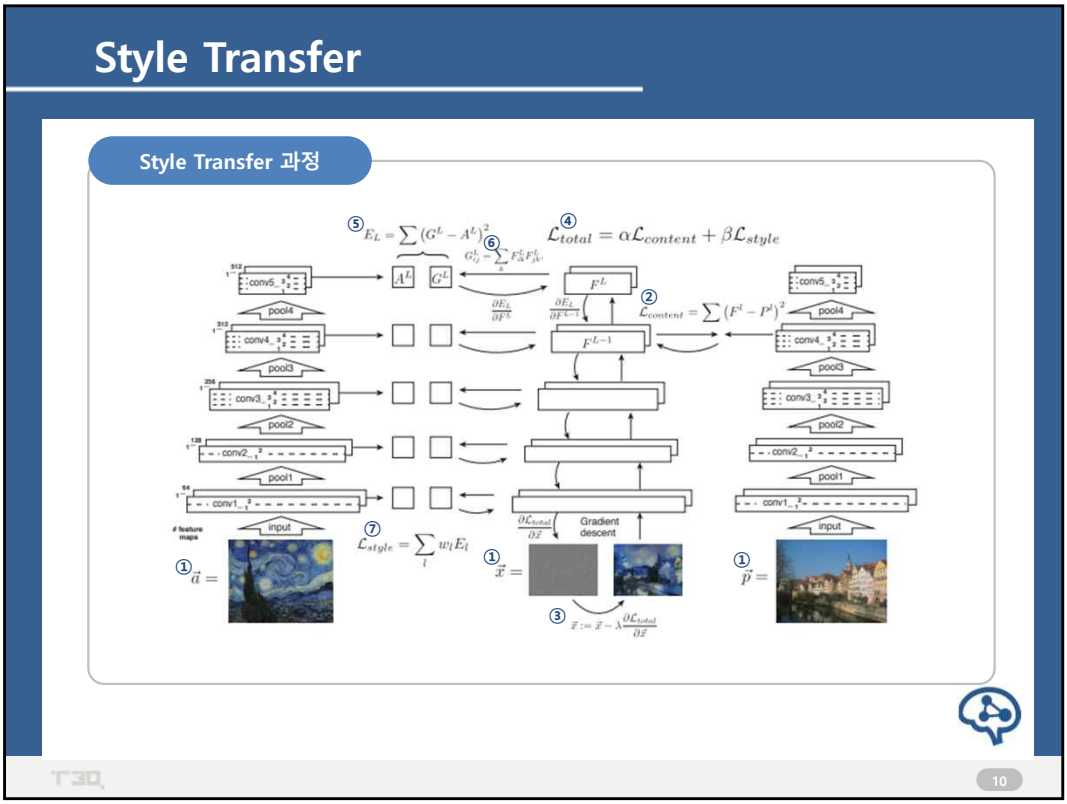
수식 값 설명

G : 특징들 간의 상관관계 값

l : layer

F : Feature map

i, j : Index 값



이미지 출처 : Image Style Transfer Using Convolutional Neural Networks(CVPR 2016) 논문

① $\vec{a}$: style image, $\vec{x}$: $\vec{a}$ 와 $\vec{p}$ 의 특징을 가져와 업데이트되는 이미지, $\vec{p}$: content image
② F^l: 업데이트 하고있는 $\vec{x}$ 의 output feature 값, P^l : $\vec{p}$ 의 output feature 값
- content loss : 한 장의 content image를 vgg19 네트워크에 넣어서 여러 개의 layer를 거쳐 conv4 layer에서 나온 output feature값 확인
- 업데이트 하고 있는 이미지 $\vec{x}$ 의 output feature와 비교해서 서로 일치할 수 있는 방향으로 업데이트 할 수 있도록 loss값 설정

③ 이미지 $\vec{p}$ 로 미분해서 얻은 Gradient 값으로 조금씩 업데이트를 반복해서 결과 이미지를 만든다.
④ L_total : content loss + style loss, $L\_content$: content loss, L_style : style loss
α, β : 가중치
⑤ G^L : 업데이트 중인 $\vec{x}$ 로부터 얻은 Gram Matrix, A^L : style image로부터 얻은 Gram Matrix
- style loss 는 여러 개의 layer를 사용하기 때문에 각각의 layer에 대한 loss값을 E_L 로 정의

⑥ 각각의 feature map에 대해서 Gram Matrix를 구해서 Gram Matrix가 일치할 수 있도록 업데이트 하는 방향으로 loss값 구함
→ 두 이미지의 스타일이 유사해질 수 있도록 만드는 것
⑦ w_l : l에 따른 가중치
- 전체 style loss는 개별적인 layer에 대해서 각각의 Loss값을 가중치를 부여해서 모두 더한 값

Style transfer 모델에서 이미지 $\vec{x}$ 는 content loss와 style loss를 모두 줄일 수 있는 방향으로 업데이트 된다.

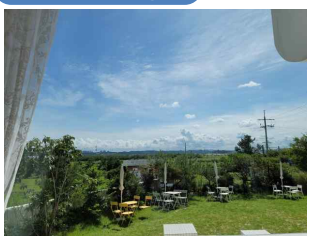
이미지 출처 : Image Style Transfer Using Convolutional Neural Networks(CVPR 2016) 논문

Style Transfer Algorithm

- 먼저 content feature, style feature를 추출하여 저장한다.
- Style image는 네트워크를 통해 전달되고 포함된 모든 레이어의 style 정보가 계산되고 저장된다.
- Content image는 네트워크를 통해 전달되고 한 레이어의 content 정보가 저장된다.
- 그런 다음 임의의 백색잡음 이미지인 x가 네트워크를 통해 전달되고 style feature(G), content feature(F)가 계산된다.
- Style feature를 추출하기 위해 각 레이어에서 G와 A 사이의 평균 제곱 차를 계산하여 Lstyle을 제공한다.
- 또한 F와 P 사이의 평균 제곱 차이를 계산하여 Lcontent를 제공한다.
- 그러면 Ltotal의 값은 Lstyle값과 Lcontent값의 합으로 계산된다.
- 스타일 이미지의 스타일 특징과 콘텐츠 이미지의 콘텐츠 특징이 동시에 일치할 때까지 이미지x를 업데이트한다.

추론

Content Image

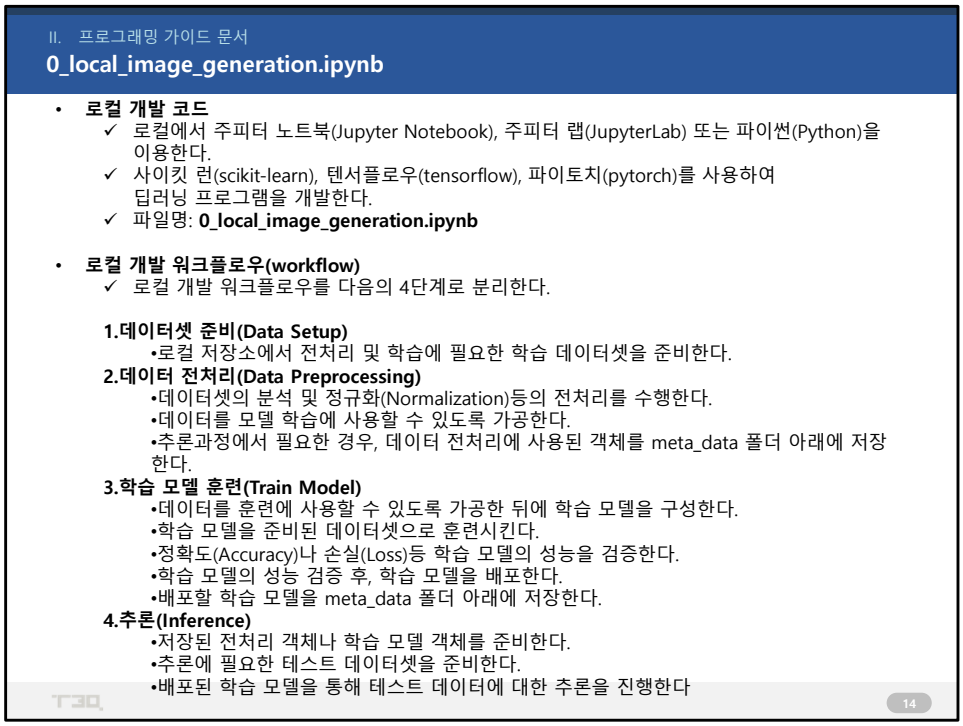


Style Image

Generate Image

0_local_image_generation_requirement

- 로컬에 설치 되어야 할 패키지 버전
  - ✓ matplotlib 3.4.3
  - ✓ tensorflow 2.8.0
  - ✓ tensorflow-gpu 2.6.0
  - ✓ pillow 8.3.2
  - ✓ numpy 1.22.2



0_local_image_generation.ipynb

```
==========================================
# imports
import os
import numpy as np
import zipfile
from glob import glob
import tensorflow as tf
from tensorflow.python.keras import models
from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
import matplotlib.pyplot as plt
from PIL import Image
import imageio

# Content feature을 뽑을 층
content_layers = ['block5_conv2']
# Style feature을 뽑을 층
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1'
               ]
num_content_layers = len(content_layers)
num_style_layers = len(style_layers)
zip_target_path = './meta_data'
os.makedirs(zip_target_path, exist_ok=True)
# 이미지 경로
content_path = zip_target_path + '/dataset/view.jpg'
```

```
style_path = zip_target_path + '/dataset/mountain.jpg'
```

1. 데이터셋 준비(Data Setup)

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

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

extract_zip_file.close()

# 데이터 확인
plt.figure(figsize = (16, 8))

content_image = Image.open(content_path)
plt.subplot(1,2,1)
plt.imshow(content_image)
plt.title('CONTENT IMAGE')
plt.axis('off')

style_image = Image.open(style_path)
plt.subplot(1,2,2)
plt.imshow(style_image)
plt.title('STYLE IMAGE')
plt.axis('off')

plt.show()
```

2. 데이터 전처리(Data Preprocessing)

```
# 스타일 전이 전처리
# 이미지 크기 조정, 형식 변환
def preprocessing_style_transfer(file):
    img = Image.open(file)
    img = img.resize((int(img.size[0]*0.5),int(img.size[1]*0.5)), Image.ANTIALIAS)
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array,axis=0)
    input_img = preprocess_input(img_array) #prepare image for model vgg19
    return input_img
```

3. 학습 모델 훈련(Train Model)

vgg19는 이미지 분류 네트워크 중 하나이다. 이 사전 훈련된 이미지 분류 네트워크를 이용하여 콘텐츠 및 스타일 표현을 정의한다.

콘텐츠 및 스타일 표현 정의

- 사전 훈련된 vgg19 네트워크 아키텍처의 중간 레이어를 이용한다.

분류 네트워크의 중간 출력을 통해 스타일 및 콘텐츠 표현을 정의할 수 있는 이유

- 높은 수준에서 이 현상을 설명할 때, 이미지 분류를 수행하려면 네트워크가 이미지를 이해해야 한다는 사실로 설명할 수 있다.
- 이미지 분류시 배경 잡음 및 기타 방해 요소에 구애받지 않는 객체(예: 개, 고양이)에서 불변성을 포착하여 분류한다.
- 그러므로 원시 이미지가 입력되는 위치와 분류 결과가 출력되는 위치 사이 어딘가에서 네트워크(vgg19)는 복잡한 특징 추출기 역할을 할 수 있다.
- 따라서 중간 레이어에 엑세스하여 입력 이미지의 내용과 스타일을 설명할 수 있다.

VGG19
- vgg19는 ResNet, Inception등과 비교했을때 상대적으로 간단한 모델이기 때문에 실제로 스타일 전송에 더 잘 작동한다.
- 이전에 선언해둔 layer 6개{5(스타일 레이어) + 1(콘텐츠 레이어)}의 특성을 출력으로 하는 모델을 생성한다.

```
def get_model():
    vgg = VGG19(include_top=False, weights='imagenet')
    vgg.trainable = False

    style_outputs = [vgg.get_layer(name).output for name in style_layers]
    content_outputs = [vgg.get_layer(name).output for name in content_layers]
    model_outputs = style_outputs + content_outputs

    return models.Model(vgg.input, model_outputs)
```

콘텐츠 손실 계산
- 생성한 모델의 중간 레이어 출력을 반환받는다.
- 입력받은 두 이미지의 중간 표현 사이의 유클리드 거리를 취한다.
- 이 콘텐츠 손실을 최소화 하기 위하여 일반적인 방식으로 역전파를 수행한다.
- 따라서 특정 레이어(content_layer)에서 원본 콘텐츠 이미지와 유사한 응답을 생성할 때까지 초기 이미지를 변경한다.

```
def get_content_loss(base_content, target):
    return tf.reduce_mean(tf.square(base_content - target))
```

스타일 손실 계산
- 기본 입력 이미지와 스타일 이미지를 네트워크에 제공한다.
- 그러나 기본 입력 이미지와 스타일 이미지의 원시 중간 출력을 비교하는 대신 두 출력의 gram matrix를 비교한다.
- 수학적으로 기본 입력 이미지 x의 스타일 손실과 스타일 이미지 a를 이러한 이미지의 스타일 표현(gram matrix) 사이의 거리로 설명한다.

```
# gram matrix 계산 함수
def gram_matrix(input_tensor):
    # We make the image channels first
    channels = int(input_tensor.shape[-1])
    a = tf.reshape(input_tensor, [-1, channels])
    n = tf.shape(a)[0]
    gram = tf.matmul(a, a, transpose_a=True)
    return gram / tf.cast(n, tf.float32)

# 스타일 손실 계산
def get_style_loss(base_style, gram_target):
    height, width, channels = base_style.get_shape().as_list()
    gram_style = gram_matrix(base_style)
    return tf.reduce_mean(tf.square(gram_style - gram_target))
```

손실 및 가중치 계산

```
# 이미지를 로드하고 네트워크를 통해 전달하는 함수를 정의한 다음
# 모델에서 콘텐츠 및 스타일 특징 표현을 출력한다.
def get_feature_representations(model, content_path, style_path):

    content_image = preprocessing_style_transfer(content_path)
    style_image = preprocessing_style_transfer(style_path)

    # 원하는 콘텐츠 이미지와 기본 입력 이미지를 네트워크에 전달한다.
    # 이것은 모델의 중간에서 선택한 레이어 출력을 반환한다.
    style_outputs = model(style_image)
    content_outputs = model(content_image)

    # 모델의 출력(style_outputs 및 content_outputs)은
    # style_feature를 추출할 출력 5개 + content_feature를 추출할 출력 1개로
    # 이루어져 있으므로 이를 스타일 특징, 콘텐츠 특징으로 반환한다.
    style_features = [style_layer[0] for style_layer in style_outputs[:num_style_layers]]
    content_features = [content_layer[0] for content_layer in content_outputs[num_style_layers:]]
    return style_features, content_features

# 손실 계산
def compute_loss(model, loss_weights, init_image, gram_style_features, content_features):
    style_weight, content_weight = loss_weights

    model_outputs = model(init_image)

    style_output_features = model_outputs[:num_style_layers]
    content_output_features = model_outputs[num_style_layers:]

    style_score = 0
    content_score = 0

    weight_per_style_layer = 1.0 / float(num_style_layers)
    for target_style, comb_style in zip(gram_style_features, style_output_features):
        style_score += weight_per_style_layer * get_style_loss(comb_style[0], target_style)

    weight_per_content_layer = 1.0 / float(num_content_layers)
    for target_content, comb_content in zip(content_features, content_output_features):
        content_score += weight_per_content_layer * get_content_loss(comb_content[0],
target_content)

    style_score *= style_weight
    content_score *= content_weight

    loss = style_score + content_score
    return loss, style_score, content_score

# 가중치 계산
def compute_grads(cfg):
    with tf.GradientTape() as tape:
        all_loss = compute_loss(**cfg)

    total_loss = all_loss[0]
```

```
return tape.gradient(total_loss, cfg['init_image']), all_loss
```

```
# 가중치와 손실을 계산하는 과정에서 norm_means를 적용시켰던 데이터를 이미지로 변환한다.
def deprocess_img(processed_img):
    x = processed_img.copy()
    if len(x.shape) == 4:
        x = np.squeeze(x, 0)
    assert len(x.shape) == 3, ("Input to deprocess image must be an image of "
                               "dimension [1, height, width, channel] or [height, width, channel]")
    if len(x.shape) != 3:
        raise ValueError("Invalid input to deprocessing image")
    # perform the inverse of the preprocessing step
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    x = x[:, :, ::-1]

    x = np.clip(x, 0, 255).astype('uint8')
    return x

def run_style_transfer(content_path,
                       style_path,
                       num_iterations=1000,
                       content_weight=1e3,
                       style_weight=1e-2):
    model = get_model()
    for layer in model.layers:
        layer.trainable = False

    # style_features 5개, content_features 1개
    style_features, content_features = get_feature_representations(model, content_path, style_path)
    gram_style_features = [gram_matrix(style_feature) for style_feature in style_features]

    init_image = preprocessing_style_transfer(content_path)
    init_image = tf.Variable(init_image, dtype=tf.float32)

    opt = tf.optimizers.Adam(learning_rate=5, beta_1=0.99, epsilon=1e-1)

    best_loss, best_img = float('inf'), None

    loss_weights = (style_weight, content_weight)
    cfg = {
        'model': model,
        'loss_weights': loss_weights,
        'init_image': init_image,
        'gram_style_features': gram_style_features,
        'content_features': content_features
    }

    # norm_means는 불러온 모델(VGG19에 imageNet가중치)에서 학습시켰던 이미지의 채널별 픽셀
평균값이다.
    # 평균을 알면 모든 픽셀 값에서 빼서 중심이 0에 오도록 할 수 있다.
    # 이는 훈련 속도와 정확도를 높이는데 도움이 된다.
    norm_means = np.array([103.939, 116.779, 123.68])
    min_vals = -norm_means
    max_vals = 255 - norm_means
```

```
    # 생성한 이미지 저장
    generated_images = np.expand_dims(deprocess_img(init_image.numpy()), axis=0)
    losses = []

    for i in range(1, num_iterations+1):
        grads, all_loss = compute_grads(cfg)
        loss, style_score, content_score = all_loss
        opt.apply_gradients([(grads, init_image)])
        # 범위를 벗어나는 값은 버린다.
        clipped = tf.clip_by_value(init_image, min_vals, max_vals)
        init_image.assign(clipped)
        if i % 10 == 0:
            image = np.expand_dims(deprocess_img(init_image.numpy()), axis=0)
            generated_images = np.concatenate((generated_images, image), axis=0)
        losses.append(loss.numpy())

    return generated_images, losses

# 상단 과정 함수화하여 실행
images, losses = run_style_transfer(content_path,
                                    style_path, num_iterations=1000)

# 변화과정 시각화
gif_dir = os.path.join('./meta_data', 'generate_images.gif')
imageio.mimsave(gif_dir, images, 'GIF', fps=10)

plt.plot(losses)
plt.title('LOSS')
plt.show()

# style을 적용시킨 최종 content이미지
final_content = Image.fromarray(images[-1])
final_content
final_content.save('./meta_data/[0_local]view.jpg')
```

4. 추론(Inference)
해당 모델의 추론은 **content**이미지만 변경하여 다시 학습한다.

```
zip_test_target_path = './meta_data'
os.makedirs(zip_test_target_path, exist_ok=True)
# test_dataset.zip 파일을 test_dataset 폴더에 압축을 풀어준다.
zip_test_source_path = './test_dataset.zip'

extract_zip_file = zipfile.ZipFile(zip_test_source_path)
extract_zip_file.extractall(zip_test_target_path)

extract_zip_file.close()
content_path = glob(os.path.join(zip_test_target_path, 'test_dataset/*.jpg'))[-1]
# 이미지 생성
images, losses = run_style_transfer(content_path,
                                    style_path, num_iterations=1000)
# 생성한 이미지 확인
inference_content = Image.fromarray(images[-1])
inference_content
```

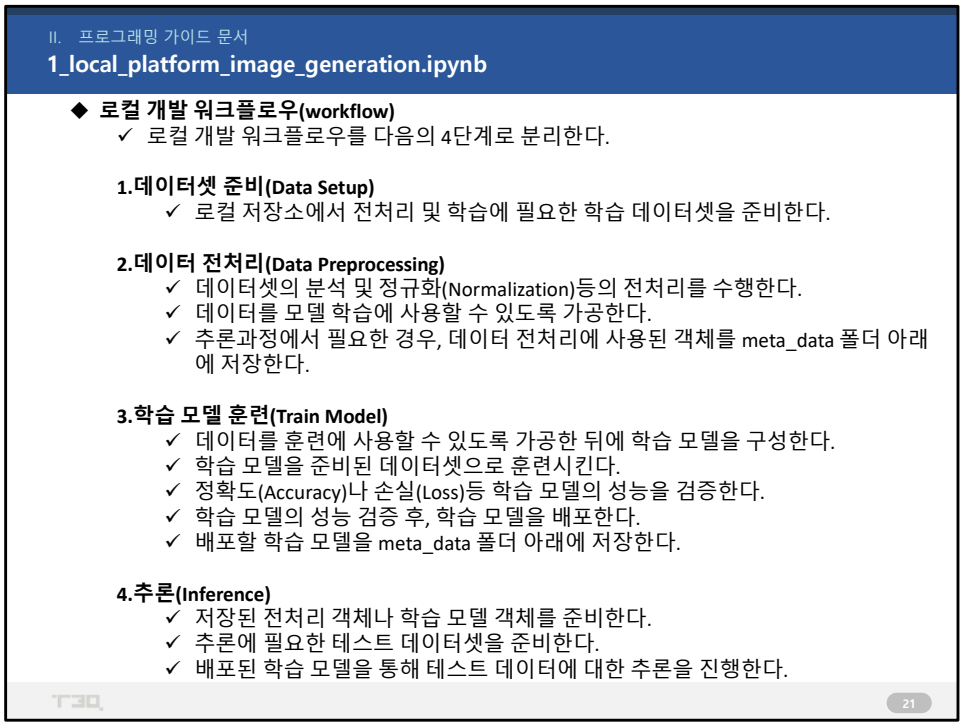
```
inference_content.save('./meta_data/[0_local]river.jpg')
```

1_local_platform_image_generation.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_image_generation.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: image_generation_preprocess.py

'''
from image_generation_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: image_generation_preprocess_sub.py

import os
import numpy as np
import pandas as pd
import zipfile
import logging


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))

    # 저장 파일 확인
    list_files_directories(pm.source_path)

    # pm.source_path의 dataset.zip 파일을
    # pm.target_path의 dataset 폴더에 압축을 풀어준다.
    my_zip_path = os.path.join(pm.source_path,'dataset.zip')
    extract_zip_file = zipfile.ZipFile(my_zip_path)
    extract_zip_file.extractall(pm.target_path)
    extract_zip_file.close()

    # 저장 파일 확인
```

```
    list_files_directories(pm.target_path)

    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: image_generation_train.py

'''
from image_generation_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: image_generation_train_sub.py

# imports
import os
import io
import shutil
import numpy as np
import base64

import tensorflow as tf
from tensorflow.python.keras import models
from tensorflow.keras.applications.vgg19 import VGG19, preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array

from PIL import Image
import imageio
import logging

logging.getLogger('PIL').setLevel(logging.WARNING)

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

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

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)

# Content feature을 뽑을 층
content_layers = ['block5_conv2']

# Style feature을 뽑을 층
style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1'
               ]

num_content_layers = len(content_layers)
num_style_layers = len(style_layers)

def exec_train(tm):

    logging.info('[hunmin log] the start line of the function [exec_train]')
    logging.info('[hunmin log] tm.train_data_path : {}'.format(tm.train_data_path))

    # 저장 파일 확인
    list_files_directories(tm.train_data_path)
```

```
    ##################################################################
##########
    ## 1. 데이터 세트 준비(Data Setup)
    ##################################################################
##########

    data_path = tm.train_data_path + '/dataset'
    list_files_directories(data_path)

    # 이미지 경로
    content_path = data_path + '/view.jpg'
    style_path = data_path + '/mountain.jpg'

    logging.info('[hunmin log] content_path : {}'.format(content_path))
    logging.info('[hunmin log] style_path : {}'.format(style_path))

    # 스타일 이미지 저장
    style_save_path = os.path.join(tm.model_path, 'mountain.jpg')
    shutil.copyfile(style_path, style_save_path)

    list_files_directories(tm.model_path)

    # 스타일 전이 수행
    images, losses = run_style_transfer(content_path,
                                        style_path, num_iterations=1000)
```

```
    logging.info('[hunmin log] losses : {}'.format(losses))

    ##################################################################
##########
    ## 플랫폼 시각화
    ##################################################################
##########
    '''
    plot_metrics(tm, losses)
    '''

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


def exec_init_svc(im):
    style_path = os.path.join(im.model_path, 'mountain.jpg')
    return { 'style_path': style_path }


def exec_inference(df, params, batch_id):
```

```
    ##################################################################
##########
    ## 4. 추론(Inference)
    ##################################################################
##########

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

    # 스타일 이미지 경로 불러오기
    style_path = params['style_path']
    logging.info('[hunmin log] style_path : {}'.format(style_path))

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

    # compute grads & loss
    images, losses = run_style_transfer(image_bytes, style_path)

    # # save image
    # target_path = os.path.join(hunmin_dir, 'TRANSFORM.jpg')
    # Image.fromarray(images[-1]).save(target_path)

    # inverse transform
    result = {'inference' : "Style Transfer Complete"}
    logging.info('[hunmin log] result : {}'.format(result))

    return result



# 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))

##################################################################
## exec_train(tm) 호출 함수
##################################################################

##################################################################
## 2. 데이터 전처리(Data Preprocessing)
##################################################################

# 스타일 전이 전처리
# 이미지 크기 조정, 형식 변환
def preprocessing_style_transfer(file):
    img = Image.open(file)
    img = img.resize((int(img.size[0]*0.5),int(img.size[1]*0.5)), Image.ANTIALIAS)
    img_array = img_to_array(img)
    img_array = np.expand_dims(img_array,axis=0)
    input_img = preprocess_input(img_array) #prepare image for model vgg16
    return input_img
```

```
########################################################################
## 3. 학습 모델 훈련(Train Model)
########################################################################
# 모델 정의
def get_model():
    vgg = VGG19(include_top=False, weights='imagenet')
    vgg.trainable = False

    style_outputs = [vgg.get_layer(name).output for name in style_layers]
    content_outputs = [vgg.get_layer(name).output for name in content_layers]
    model_outputs = style_outputs + content_outputs

    return models.Model(vgg.input, model_outputs)

# 콘텐츠 이미지 손실 계산
def get_content_loss(base_content, target):
    return tf.reduce_mean(tf.square(base_content - target))

# gram matrix 계산 함수(스타일 이미지 손실 계산에 이용)
def gram_matrix(input_tensor):
    # We make the image channels first
    channels = int(input_tensor.shape[-1])
    a = tf.reshape(input_tensor, [-1, channels])
    n = tf.shape(a)[0]
    gram = tf.matmul(a, a, transpose_a=True)
    return gram / tf.cast(n, tf.float32)

# 스타일 손실 계산
def get_style_loss(base_style, gram_target):
    height, width, channels = base_style.get_shape().as_list()
    gram_style = gram_matrix(base_style)
    return tf.reduce_mean(tf.square(gram_style - gram_target))

# 이미지를 로드하고 네트워크를 통해 전달하는 함수를 정의한 다음
# 모델에서 콘텐츠 및 스타일 특징 표현을 출력한다.
def get_feature_representations(model, content_path, style_path):

    content_image = preprocessing_style_transfer(content_path)
    style_image = preprocessing_style_transfer(style_path)

    # 원하는 콘텐츠 이미지와 기본 입력 이미지를 네트워크에 전달한다.
    # 이것은 모델의 중간에서 선택한 레이어 출력을 반환한다.
    style_outputs = model(style_image)
    content_outputs = model(content_image)

    # 모델의 출력(style_outputs 및 content_outputs)은
    # style_feature를 추출할 출력 5개 + content_feature를 추출할 출력 1개로
    # 이루어져 있으므로 이를 스타일 특징, 콘텐츠 특징으로 반환한다.
    style_features = [style_layer[0] for style_layer in style_outputs[:num_style_layers]]
    content_features = [content_layer[0] for content_layer in
content_outputs[num_style_layers:]]
    return style_features, content_features
```

```
# 손실 계산
def compute_loss(model, loss_weights, init_image, gram_style_features, content_features):
    style_weight, content_weight = loss_weights

    model_outputs = model(init_image)

    style_output_features = model_outputs[:num_style_layers]
    content_output_features = model_outputs[num_style_layers:]

    style_score = 0
    content_score = 0

    weight_per_style_layer = 1.0 / float(num_style_layers)
    for target_style, comb_style in zip(gram_style_features, style_output_features):
        style_score += weight_per_style_layer * get_style_loss(comb_style[0], target_style)

    weight_per_content_layer = 1.0 / float(num_content_layers)
    for target_content, comb_content in zip(content_features, content_output_features):
        content_score += weight_per_content_layer * get_content_loss(comb_content[0],
target_content)

    style_score *= style_weight
    content_score *= content_weight

    loss = style_score + content_score
    return loss, style_score, content_score

# 가중치 계산
def compute_grads(cfg):
    with tf.GradientTape() as tape:
        all_loss = compute_loss(**cfg)

    total_loss = all_loss[0]
    return tape.gradient(total_loss, cfg['init_image']), all_loss

# 가중치와 손실을 계산하는 과정에서 norm_means를 적용시켰던 데이터를 이미지로 변환한
다.
def deprocess_img(processed_img):
    x = processed_img.copy()
    if len(x.shape) == 4:
        x = np.squeeze(x, 0)
    assert len(x.shape) == 3, ("Input to deprocess image must be an image of "
                               "dimension [1, height, width, channel] or [height, width, channel]")
    if len(x.shape) != 3:
        raise ValueError("Invalid input to deprocessing image")
    # perform the inverse of the preprocessing step
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    x = x[:, :, ::-1]

    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

```
# 스타일 전이 수행
def run_style_transfer(content_path,
                       style_path,
                       num_iterations=1000,
                       content_weight=1e3,
                       style_weight=1e-2):
    model = get_model()
    for layer in model.layers:
        layer.trainable = False

    # style_features 5개, content_features 1개
    style_features, content_features = get_feature_representations(model, content_path,
style_path)
    gram_style_features = [gram_matrix(style_feature) for style_feature in style_features]

    init_image = preprocessing_style_transfer(content_path)
    init_image = tf.Variable(init_image, dtype=tf.float32)

    opt = tf.optimizers.Adam(learning_rate=5, beta_1=0.99, epsilon=1e-1)

    best_loss, best_img = float('inf'), None

    loss_weights = (style_weight, content_weight)
    cfg = {
        'model': model,
        'loss_weights': loss_weights,
        'init_image': init_image,
        'gram_style_features': gram_style_features,
        'content_features': content_features
    }

    # norm_means는 불러온 모델(VGG19에 imageNet가중치)에서 학습시켰던 이미지의 채널
별 픽셀 평균값이다.
    # 평균을 알면 모든 픽셀 값에서 빼서 중심이 0에 오도록 할 수 있다.
    # 이는 훈련 속도와 정확도를 높이는데 도움이 된다.
    norm_means = np.array([103.939, 116.779, 123.68])
    min_vals = -norm_means
    max_vals = 255 - norm_means

    # 생성한 이미지 저장
    generated_images = np.expand_dims(deprocess_img(init_image.numpy()), axis=0)
    losses = []

    for i in range(1, num_iterations+1):
        grads, all_loss = compute_grads(cfg)
        loss, style_score, content_score = all_loss
        opt.apply_gradients([(grads, init_image)])
        # 범위를 벗어나는 값은 버린다.
        clipped = tf.clip_by_value(init_image, min_vals, max_vals)
        init_image.assign(clipped)
        if i % 10 == 0:
            image = np.expand_dims(deprocess_img(init_image.numpy()), axis=0)
            generated_images = np.concatenate((generated_images, image), axis=0)
        losses.append(loss.numpy())
```

```
return generated_images, losses
```

```
# 시각화
def plot_metrics(tm, losses):
    loss_list = losses
    for step, loss in enumerate(loss_list):
        metric={}
        metric['accuracy'] = 0
        metric['loss'] = loss
        metric['step'] = step
        tm.save_stat_metrics(metric)

    logging.info('[hunmin log] loss curve plot for platform')
```

```
# 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'

# 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)

# 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=[['']]

df = pd.DataFrame(data)
print('df: ', df)
print('df.dtypes:', df.dtypes)
df.columns

%%time
process_for_train(pm)
train(tm)
transform(df, params, batch_id)
params = init_svc(im)
inference(df, params, batch_id)
```

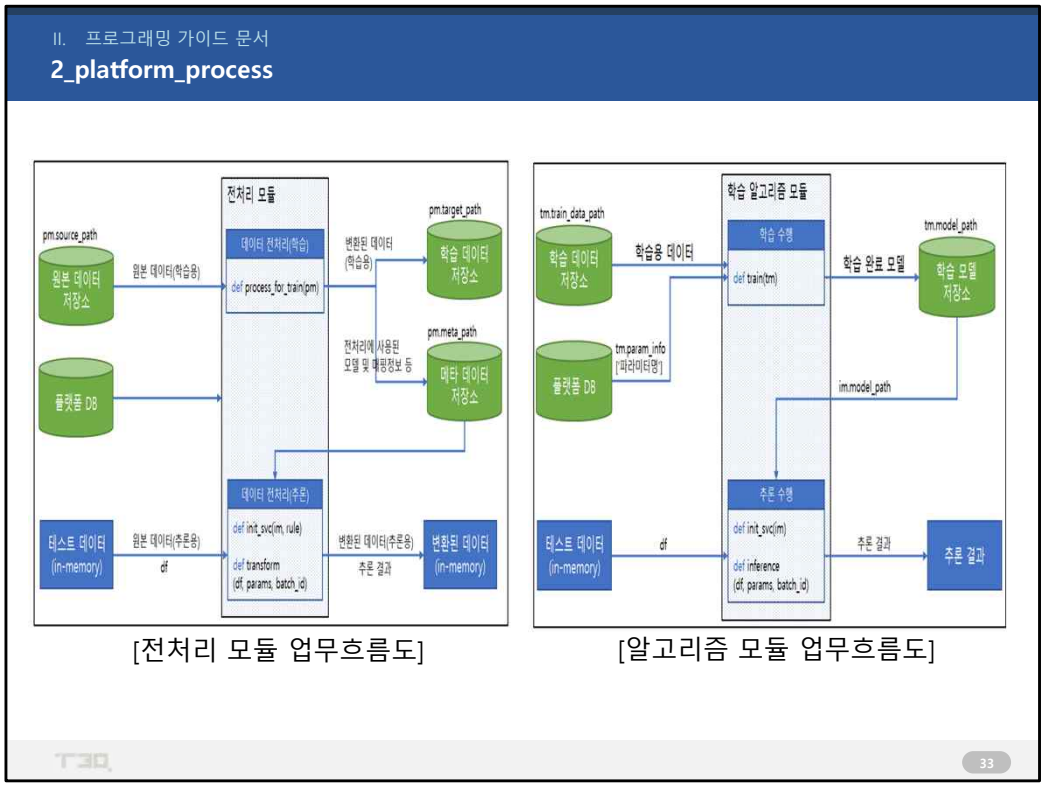
2_platform_process

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

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

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

- 파일명: image_generation_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)



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_image_generation : 해당 예제에서는 수행 절차 없음

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

03) T3Q.dl_프로젝트 설정_실행환경 관리_image_generation

04) T3Q.dl_프로젝트 설정_전처리 모듈 관리_image_generation

05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_image_generation

06) T3Q.dl_학습플랫폼_데이터셋 관리_image_generation

07) T3Q.dl_학습플랫폼_전처리 모델 설계_image_generation

08) T3Q.dl_학습플랫폼_전처리 모델 관리_image_generation

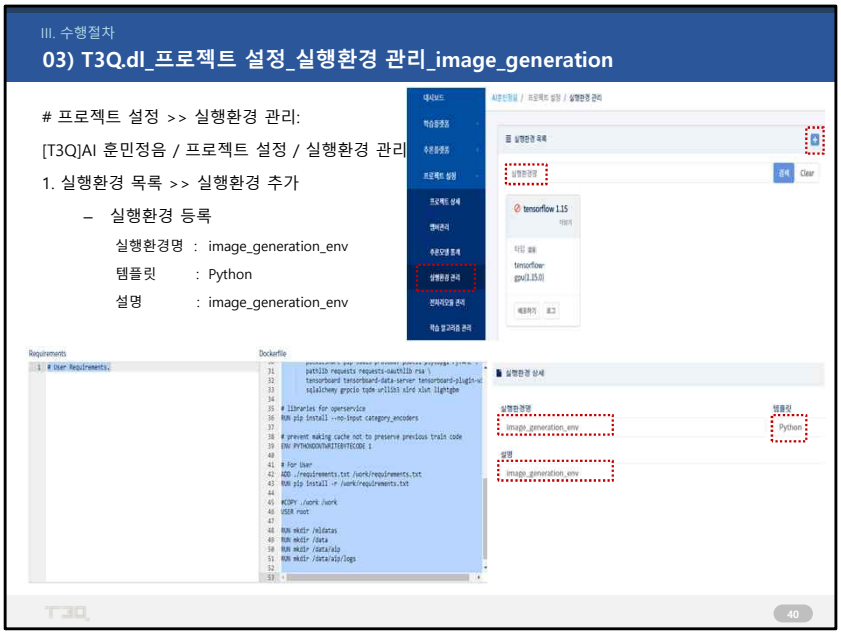
09) T3Q.dl_학습플랫폼_학습모델 설계_image_generation

10) T3Q.dl_학습플랫폼_학습모델 관리_image_generation

11) T3Q.dl_추론플랫폼_추론모델 관리_image_generation

12) T3Q.dl_추론플랫폼_추론API관리_image_generation

13) T3Q.cep_실시간 추론 파이프라인_image_generation



실행환경 추가 내용 및 절차

1) Requirements

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

2) Dockerfile

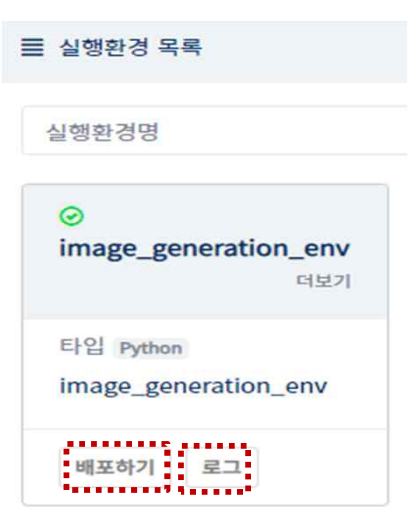
```
==================================================
FROM tensorflow/tensorflow:2.4.1-gpu

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
```

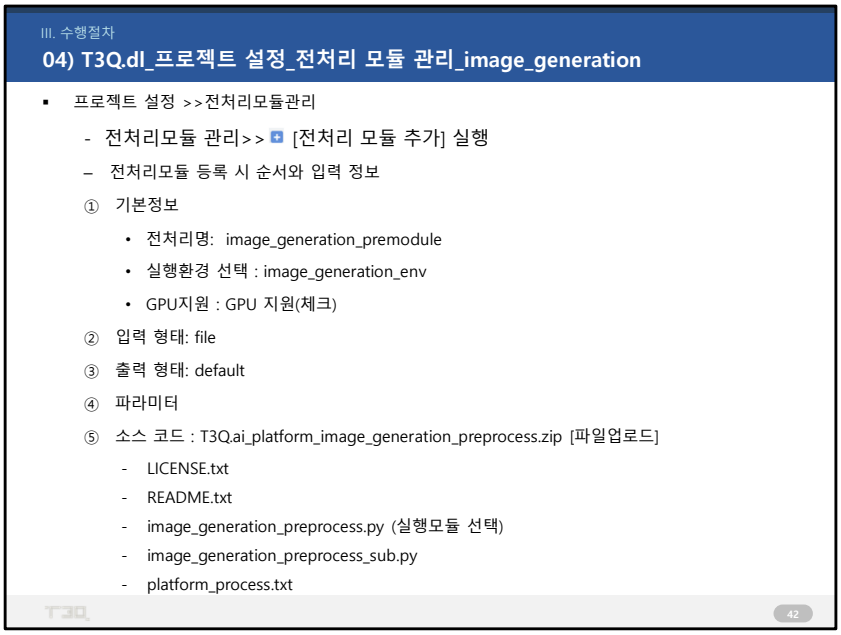
==========
2) Dockerfile-계속
==========

```
# 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

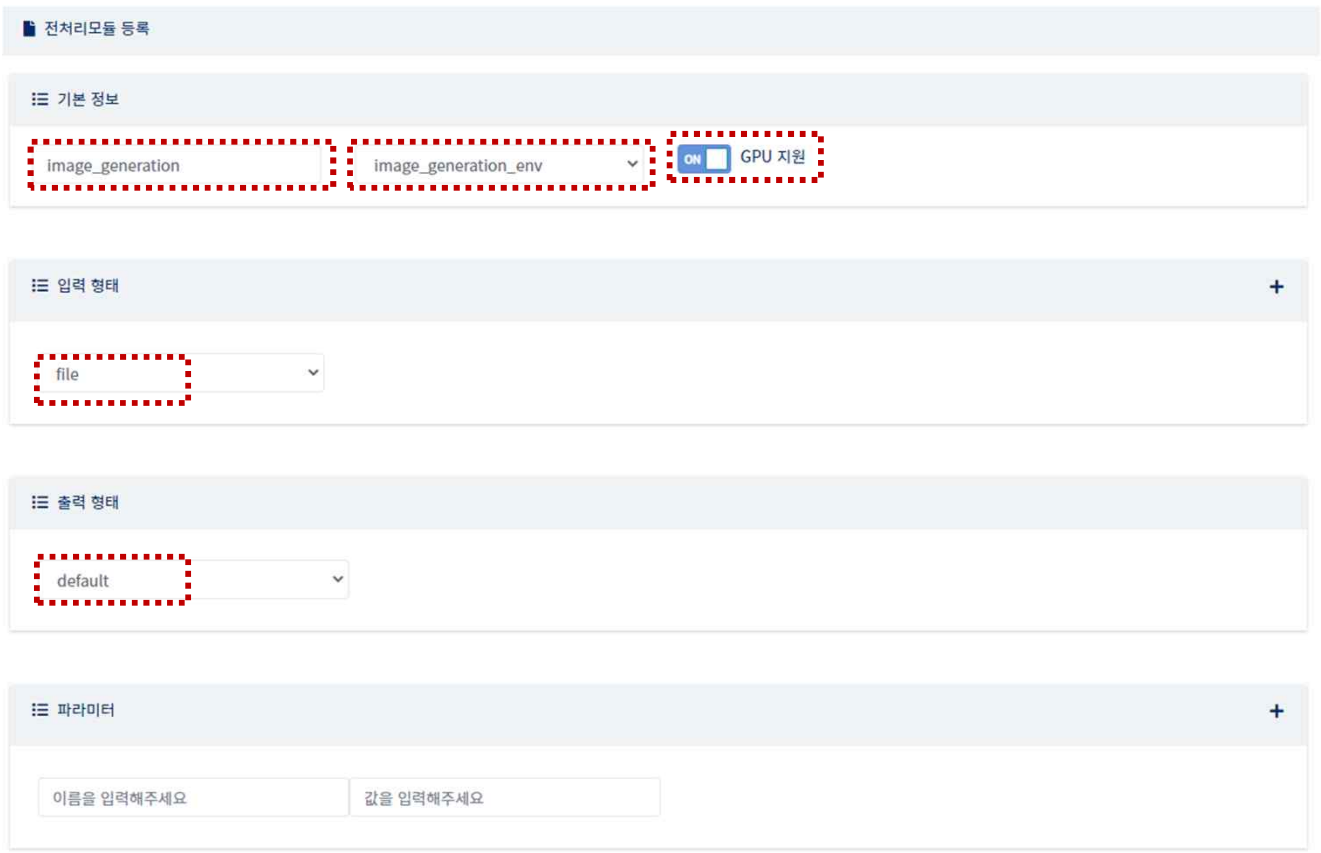
전처리명: image_generation_premodule

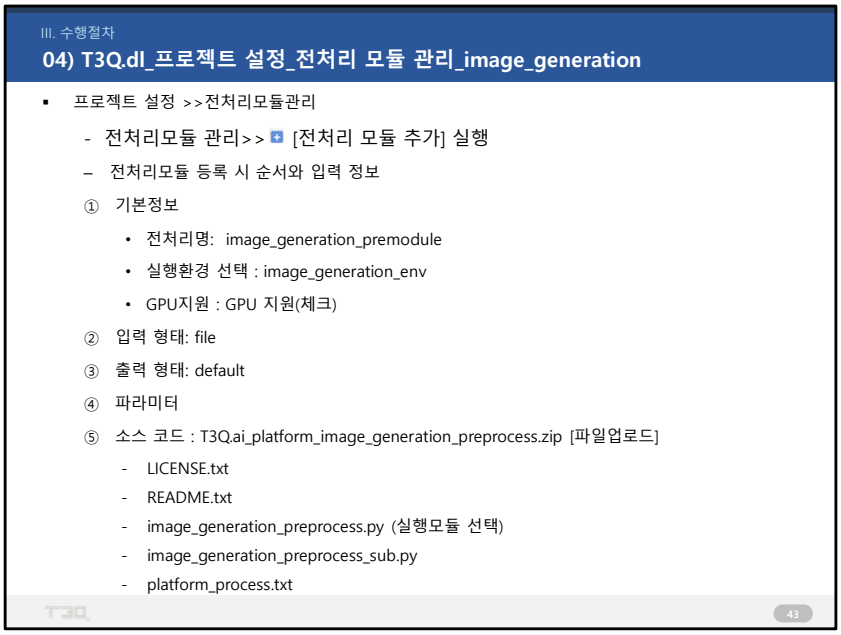
실행환경 선택 : image_generation_env

GPU지원 : GPU 지원(체크)

② 입력 형태: file

③ 출력 형태: default

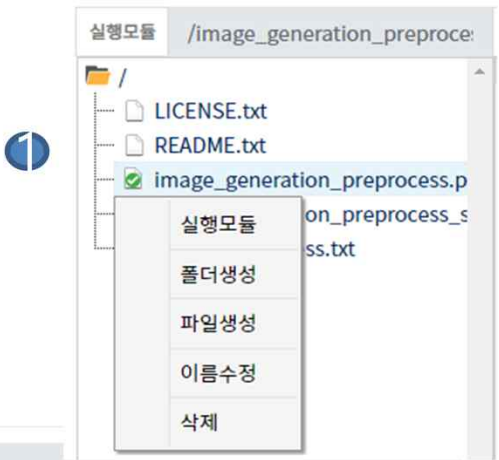




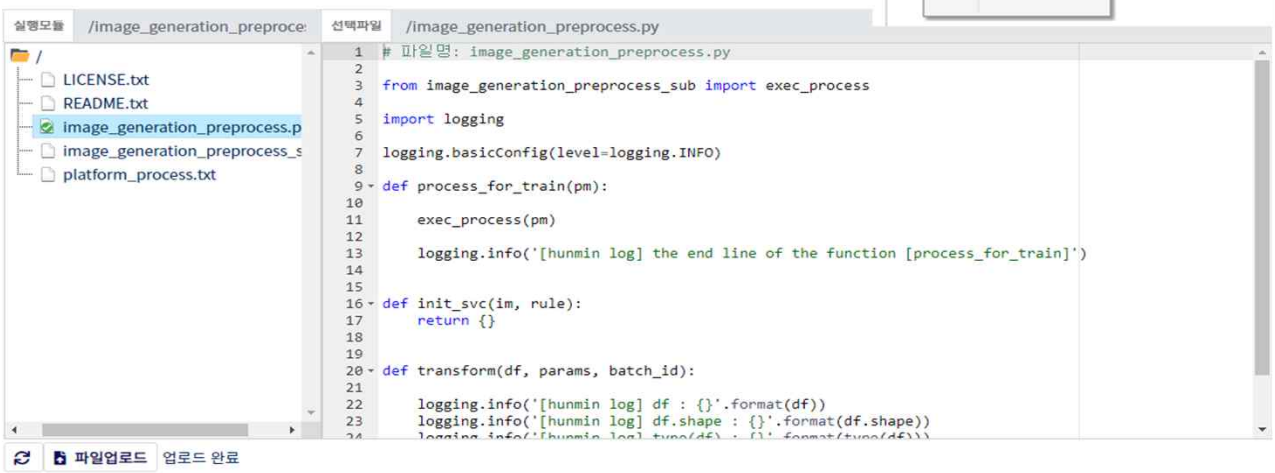
전처리모듈 등록 시 순서와 입력 정보

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

- LICENSE.txt
- README.txt
- image_generation_preprocess.py (실행모듈 선택)
- image_generation_preprocess_sub.py
- platform_process.txt



② 소스 코드



05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_image_generation

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

- 학습 알고리즘 등록 시 순서와 입력 정보

① 기본정보: 다음과 같이 입력
-알고리즘명: image_generation_train
-설명 : image_generation_train
-카테고리 : Transform
-실행환경 : image_generation_env
-GPU 지원 : 체크

② 공통 파라미터: 아래 1가지 항목 사용
- 초기화 방법: 사용

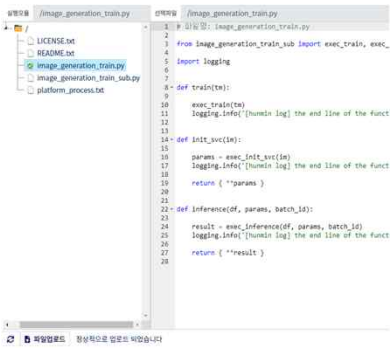
③ 모델 파라미터

④ 시각화 설정: 아래 1개 항목만 체크
- Loss : 체크

05) T3Q.dl_프로젝트 설정_학습 알고리즘 관리_image_generation

- 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_image_generation_train.zip

- LICENSE.txt
- README.txt
- image_generation_train.py
- image_generation_train_sub.py
- platform_process.txt

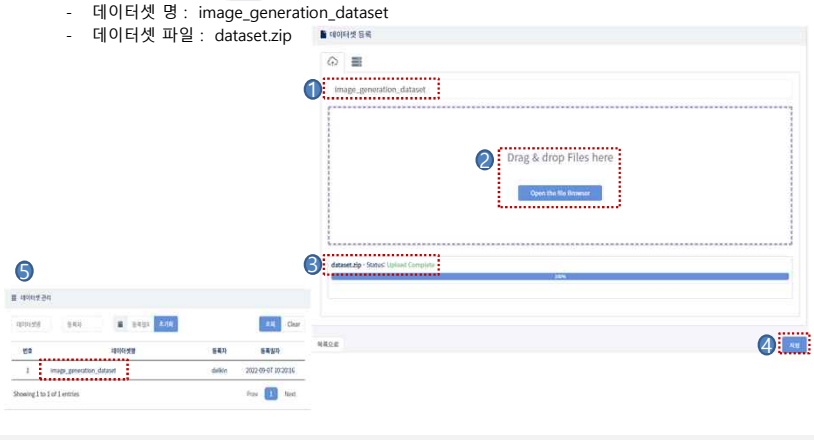
: 실행 모듈 설정 >> [저장]

- /image_generation_train.py 실행 모듈 지정
- [저장]을 누른다.

III. 수행절차

06) T3Q.dl_학습플랫폼_데이터셋 관리_image_generation

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



III. 수행절차

07) T3Q.dl_학습플랫폼_전처리 모델 설계_image_generation

- 학습플랫폼 >> 전처리 모델 설계
- 전처리 모델 설계 절차

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

② image_generation_dataset 아래의
[전처리 모델 설계] 선택

07) T3Q.dl_학습플랫폼_전처리 모델 설계_image_generation

- 학습플랫폼 >> 전처리 모델 설계

❷ 전처리 모델 설계 절차

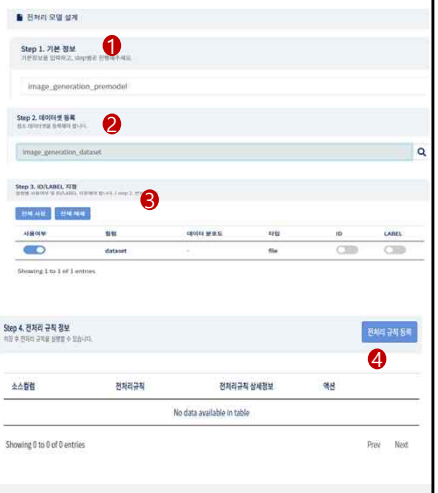
Step1. 기본 정보
- 모델명: image_generation_premodel

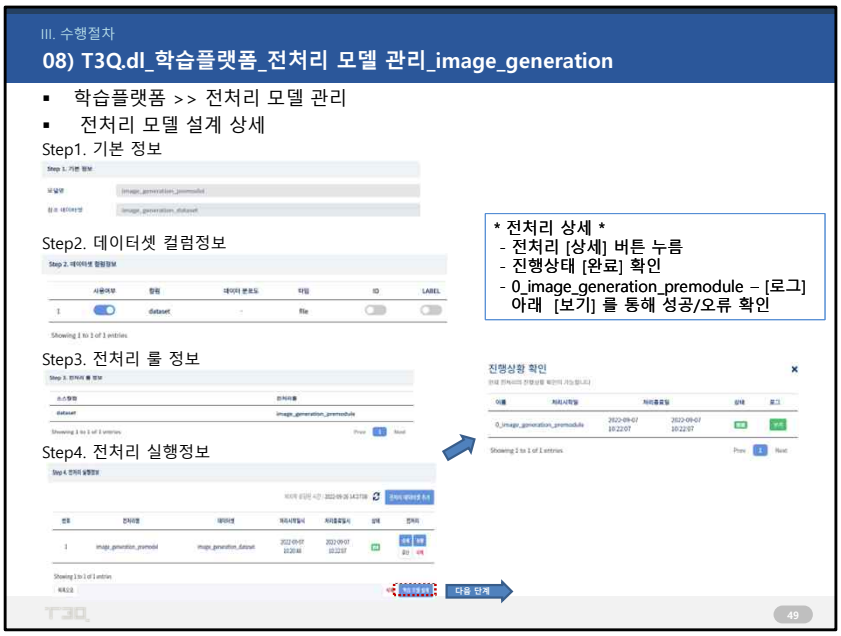
Step2. 데이터셋 등록:
- 참조데이터셋 : image_generation_dataset

Step3. ID/LABEL 지정

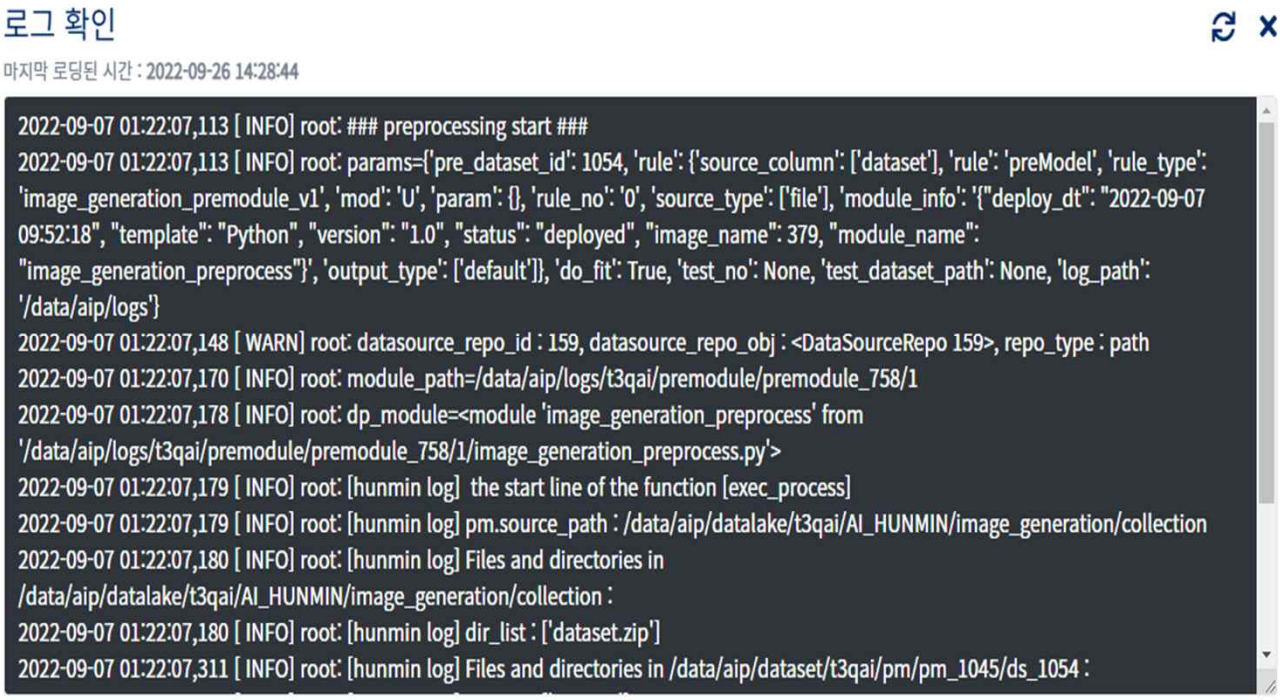
Step4. 전처리 규칙 정보
- [전처리 규칙 등록] 선택

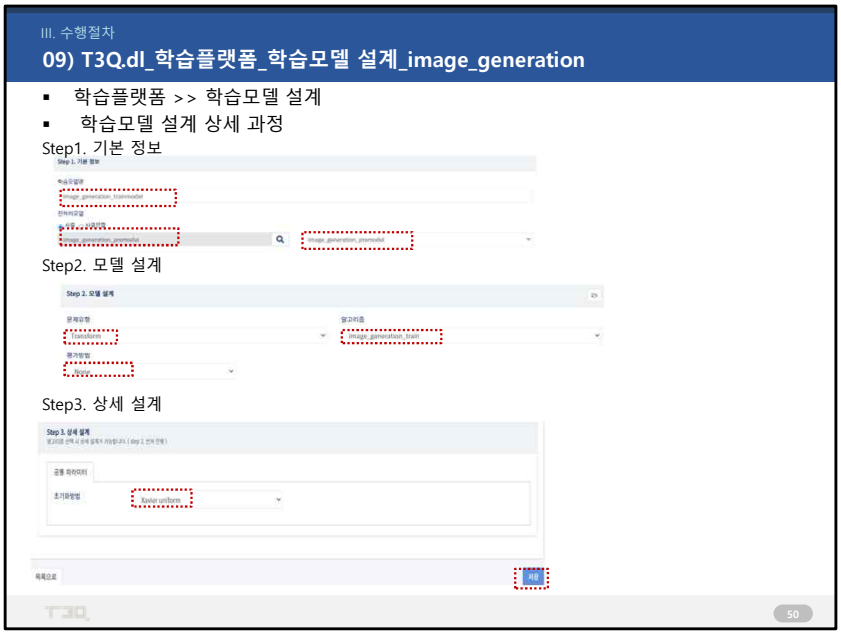
❸ 전처리 규칙 등록 시 절차
- 소스컬럼 : dataset(file) 선택
- 변환 함수 : image_generation_premodule 선택 후 [저장]





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





학습플랫폼 >> 학습모델 설계

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

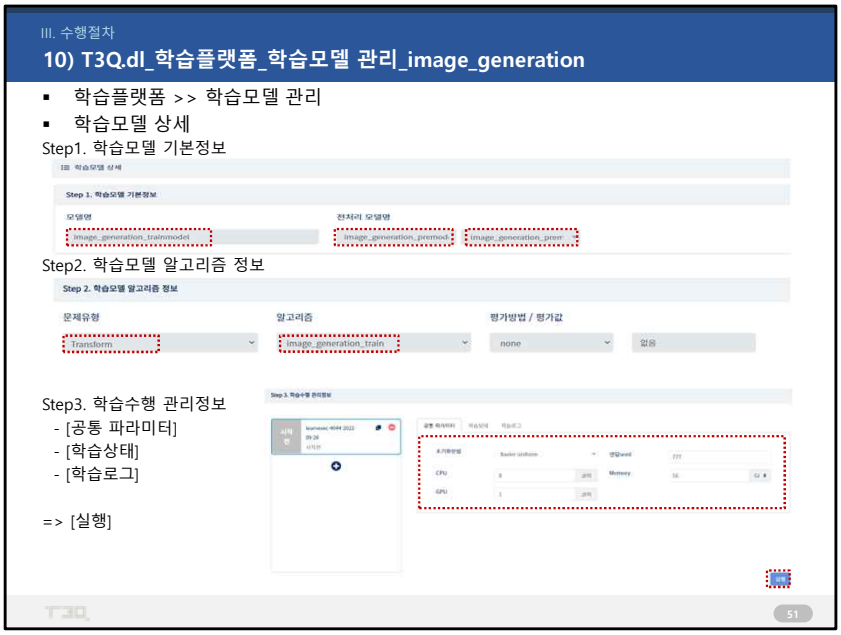
   2) Step 2. 모델 설계
      문제유형 Transform
      알고리즘 image_generation_train
      평가방법 None

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

      (1) 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 image_generation_trainmodel
전처리 모델명 image_generation_premodel image_generation_premodel

2) Step 2. 학습모델 알고리즘 정보
문제유형 Transform
알고리즘 image_generation_train
평가방법 / 평가값 None

3) Step 3. 학습수행 관리정보
(1) 공통 파라미터

초기화방법	Xavier Uniform
랜덤seed	777
CPU	8 코어
Memory	16 Gi
GPU	1 코어

(2) 학습상태
학습상태 시작전 [- ~ -]

Loss

[실행] 버튼 누름

III. 수행절차

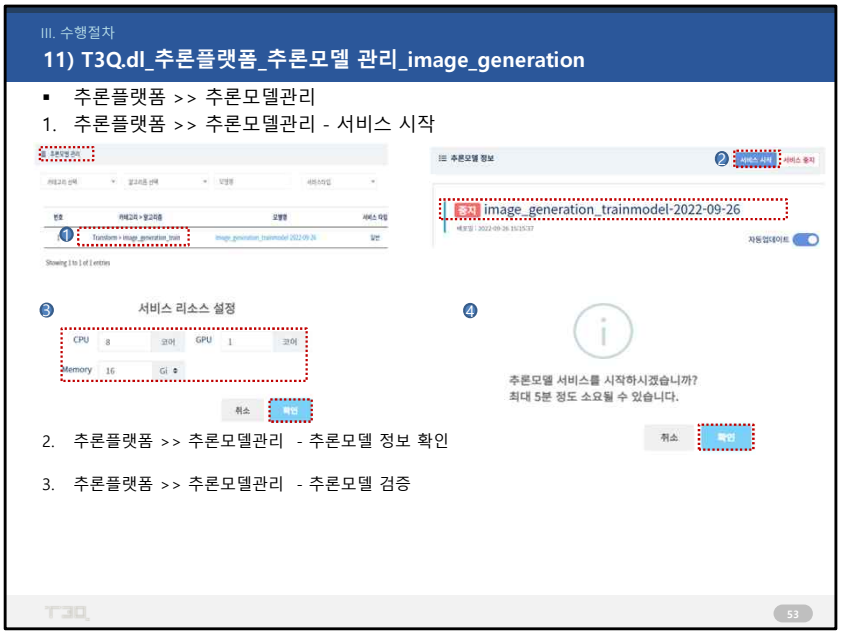
10) T3Q.dl_프로젝트 설정_학습모델 관리_image_generation

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

- [학습상태]

- [학습로그]

- 모델 배포 : [모델 배포] 버튼 누르고 [배포] 누름



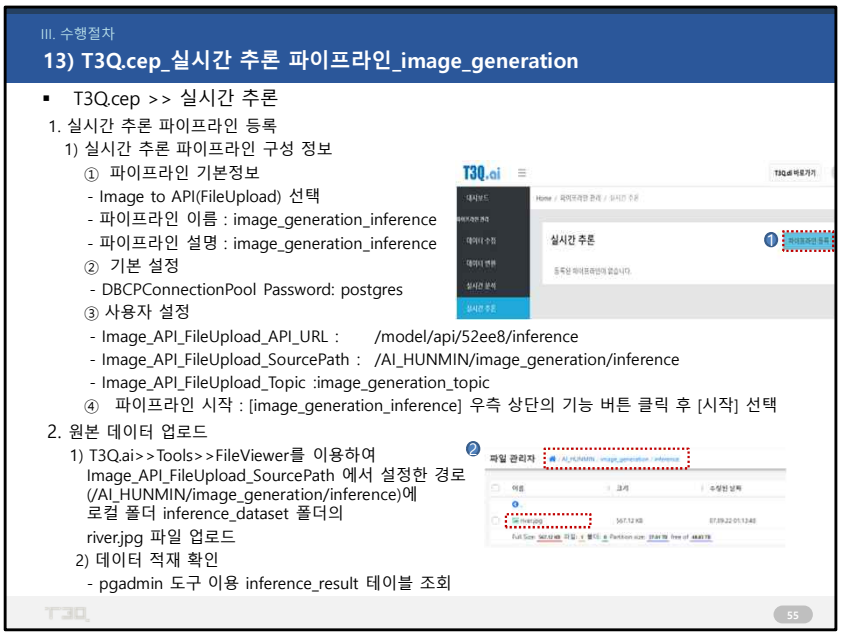
추론플랫폼 >> 추론모델관리

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_image_generation.txt' 파일 안에 있는 base64 형식의
이미지의 내용을 복사 붙여넣기 한다
응답 : "{₩"inference₩":₩"Style Transfer Complete₩"}₩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/c4047/inference%'
order by start_time desc