



Tips



[파이썬 코딩] 복사에도 종류가 있다고?

직장에서 자주 겪는 상황을 떠올려보자! AI연구센터장이 엑셀 파일을 공유하며 말한다.

<김센터장> "이 파일로 작업해 주세요. 하지만 원본은 건드리지 말고요!" (하루 뒤...)

<이대리> "헉! 보내주신 링크를 열어 직접 수정했더니, 원본 데이터까지 바뀌었어요!"

<김연구원> "이럴 줄 몰랐어요. 복사한 줄 알았는데, 원본도 망가졌어요. ㅠㅠ"

이런 상황은 실제로 많은 사람이 겪는 실수이다. 하지만 파이썬에서도 똑같은 문제가 발생할 수 있다는 걸 알고 있는가? 파이썬에서는 데이터를 복사하는 방식에 따라 원본 데이터가 바뀌거나, 완전히 독립적인 데이터를 생성할 수 있다. 바로 얇은 복사(shallow copy)와 깊은 복사(deep copy)의 차이 때문이다. 이제, 왜 이런 차이가 생기는지 파헤쳐 보고, 올바르게 사용하는 방법을 배워보자!

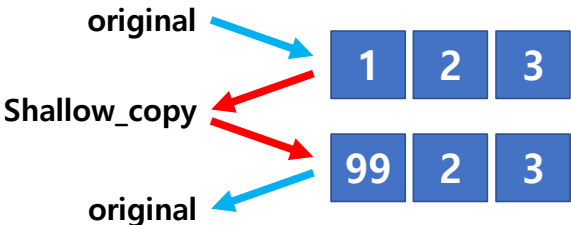
파이썬에서는 객체의 종류를 변경되는 객체인 mutable(객체의 상태를 변경할 수 있음)과 변경되지 않는 객체인 immutable(객체의 상태를 변경할 수 없음), 두 가지로 구분할 수 있다. mutable 객체로는 list, set, dictionary 정도가 있고, immutable 객체는 int, float, tuple, str, bool 등이 있다. 두 종류의 객체에 저장한 값을 호출하면 메모리를 참조하게 되는데, immutable 객체의 경우 변수에 상관없이 동일한 곳을 참조한다. 예를 들어 정수형 변수 a와 b에 3이라는 정수를 입력하면 3이라는 정수가 저장된 동일한 메모리 주소를 참조한다는 의미이다. 그러나 mutable 객체는 모든 객체값을 각각 생성해서

참조하기 때문에 list a와 b에 각각 저장된 [3]은 서로 다른 참조를 하게 된다.

문제는 후자인 mutable 객체를 다른 변수로 복사한다면, 경우에 따라 변수명만 다를 뿐 복사본과 원본의 참조가 같은 곳일 수도 있고, 새로운 참조를 하나 더 만들므로 다른 곳일 수도 있다는 것이다. 전자를 얇은복사(shallow copy), 후자를 깊은복사(deep copy)라고 한다. 이게 무슨 이야기인지 아래의 예제로 살펴보자.

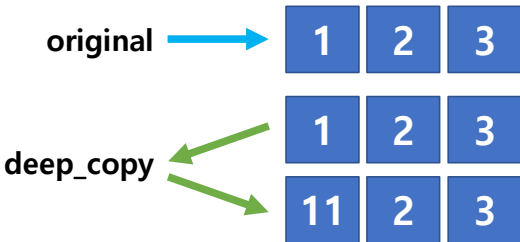
① 얇은복사(shallow copy)

얇은 복사는 복사본을 만들어도 원본 데이터와 연결된 상태이다. 즉, 복사본과 원본이 같은 참조를 공유하기 때문에, 데이터가 변경되면 두 객체에 모두 영향을 미칠 수 있다. 앞서 예로 든 상황에서 엑셀 파일의 "링크 복사"와 비슷한 경우이다. 링크를 복사하면 원본 파일에 변경이 생기면 복사한 링크에서도 똑같이 반영되게 마련이다.



② 깊은 복사(deep copy)

깊은 복사는 원본 데이터와 완전히 독립적인 복사본을 생성한다. 즉, 복사본과 원본이 서로 영향을 미치지 않는다. 앞서 예로 든 상황에서 엑셀 파일의 내용을 복사해서 새로운 빈 파일에 붙여넣기 하는 것과 같다. 원본과 복사본은 완전히 별개의 파일이 된다.



③ 언제 사용해야 할까?

얇은 복사는 간단히 구조를 복사하거나 원본 데이터를 변경할 가능성이 낮을 때 적합하다. 또한 연속된 작업을 코딩으로 구현할 때 작업 단계에 따라 변수명을 바꿀 필요에 따라 추가적인 메모리 손실 없이 사용할 수 있다. 반면에 깊은 복사는 데이터를 완전히 독립적으로 다뤄야 하거나, 중첩 구조(리스트 안에 리스트 등)에서 원본 데이터가 손상되면 안 될 때 사용하면 유용하다.

```
import copy

# 2차원 배열 생성
original = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# ① 얇은 복사
shallow_copy = original # original[:], copy.copy(original) 등
shallow_copy[0][0] = 99
print("얇은 복사본:", shallow_copy)
print("얇은 복사 후 원본:", original) # 원본도 변경됨

# ② 깊은 복사
deep_copy = copy.deepcopy(original)
deep_copy[0][0] = 11
print("깊은 복사본:", deep_copy)
print("깊은 복사 후 원본:", original) # 원본은 그대로

# 출력
얇은 복사본: [[99, 2, 3], [4, 5, 6], [7, 8, 9]]
얇은 복사 후 원본: [[99, 2, 3], [4, 5, 6], [7, 8, 9]]
깊은 복사본: [[11, 2, 3], [4, 5, 6], [7, 8, 9]]
깊은 복사 후 원본: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

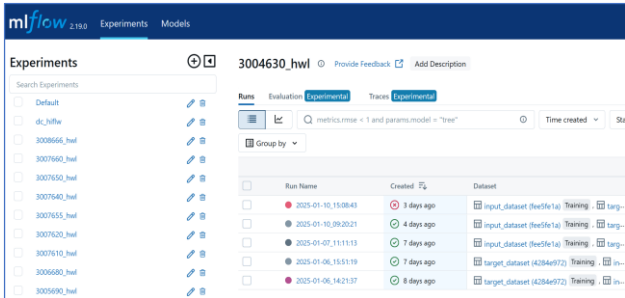
MLOps(Machine Learning & Operations)

MLOps는 기존의 소프트웨어 개발이후 배포·운영의 효율화를 위해 적용되는 DevOps(Development Operations)의 개념을 머신러닝에 적용한 것으로, 머신러닝(Machine Learning, ML)과 운영(Operaration, Ops) 프로세스를 통합하여 개발된 ML모델의 운영서비스 환경에서 모델을 안정적이고 효율적으로 배포 및 유지관리 할 수 있는 개념 및 개발을 의미한다. 따라서, ML단계에서는 데이터의 수집, 전처리, 모델 구축, 학습 및 평가 등, Ops단계에서는 모델의 배포, 모니터링 및 평가 등의 기능을 수행한다. MLOps가 적용되면 개발된 ML모델은 운영 시 효율적 버전 및 성능관리와 데이터 연계까지 하나의 프로세스로 관리가 가

능하게 되어 기존에 소프트웨어 개발자와 시스템 운영자가 분리되어 관리되던 개념을 획기적으로 개선하게 되므로 최근 다양한 분야에서 머신러닝과 딥러닝 모델의 개발 이후 효율적인 서비스 관리를 위해서 꼭 필요한 기술이다. K-water AI연구센터에서는 이러한 MLOps의 적용을 위해 미국의 빅데이터 분석기업인 데이터브릭스(Databricks)에서 오픈소스로 제공하는 MLFlow 소프트웨어를 “AI를 활용한 AI홍수분석모델”의 운영관리에 적용하기 위한 연구를 진행 중이다.



<MLFlow 적용 프로세스>



<수위유량 관측소별 딥러닝모델의 버전관리>

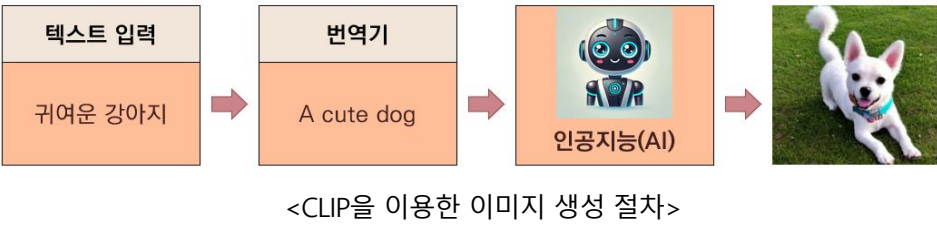


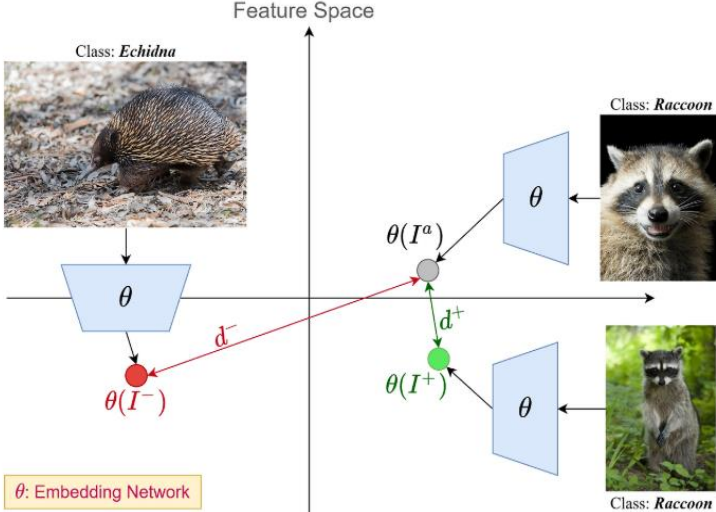
<각 관측소별 성능모니터링>



글만 쓰면 그림이 된다! CLIP

이번 Hands-On에서는 텍스트로 그림을 그려볼 것이다. OpenAI에서 개발한 CLIP이라는 모델을 사용해 한국어로 프롬프트를 입력하면 영어로 자동 번역되어 사용자가 입력한 대로 AI가 이미지를 만들어준다. 실습을 시작하기 전에 CLIP에 대해 간단하게 살펴보자. CLIP(Constrastive Language-Image Pretraining)은 텍스트와 이미지를 연결하는 멀티모달 AI이다. CLIP는 텍스트와 이미지 데이터를 함께 학습하여 텍스트-이미지 간의 관계를 이해하는데 뛰어난 성능을 보인다. CLIP의 주요 기능은 크게 ①멀티모달 학습, ②대조학습, ③제로샷 적용이며 자세한 내용은 아래 표를 참조하기 바란다. 이제 본격적인 코딩 실습으로 들어가 보자!



주요기능	설 명
멀티모달 학습	<ul style="list-style-type: none">CLIP는 텍스트와 이미지를 한 쌍으로 학습텍스트와 이미지 데이터는 각각 다른 인코더를 사용해 변환<ul style="list-style-type: none">텍스트 인코더: Transformer 기반 모델 사용이미지 인코더: CNN, ViT 사용텍스트 벡터와 이미지 벡터를 같은 공간에 매핑
대조학습 (Constrastive Learning)	<ul style="list-style-type: none">같은 쌍("고양이"와 고양이사진)은 벡터 공간상 가까워지도록, 다른 쌍("고양이"와 강아지사진)은 멀어지도록 학습 <div></div>
Zero-Shot 학습	<ul style="list-style-type: none">제로샷, 즉 해당 모델은 즉시 사용/적용이 가능합니다.예를 들어, "강아지사진" 이라는 텍스트를 입력하면, 매우 높은 확률로 바로 강아지 사진이 출력됩니다.

※ 전체 Source 코드 및 결과는 분량상 생략하며, 아래의 링크를 참조 바람
[구글 코랩에서 열기\(인터넷망\)](#)

① 관련 라이브러리 설치 및 참조

```
# 필요한 라이브러리 설치
!pip install diffusers transformers accelerate torch torchvision numpy matplotlib googletrans==4.0.0-rc1

# 해당 라이브러리에서 필요한 모듈 참조
from diffusers import StableDiffusionPipeline
from googletrans import Translator
import torch
from matplotlib import pyplot as plt
```

② GPU 선택(코랩에서 무료 지원) 및 번역기 세팅

```
# GPU가 제대로 선택되었는지 확인(cuda로 선택)
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"사용중인 device : {device}")

# Stable Diffusion 모델 로드
pipeline = StableDiffusionPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5", # 또는 다른 Stable Diffusion 모델
    torch_dtype=torch.float16)
pipeline = pipeline.to(device)

# Google Translator 설정
translator = Translator()
```

③ 텍스트(프롬프트) 입력

```
# 한국어 텍스트로 그리고 싶은 그림을 묘사하는 프롬프트 입력
Korean_prompt = " 귀여운 강아지"

# 아래는 예시
*** korean_prompt = "슈퍼맨 복장을 한 고양이"
korean_prompt = "별빛 아래에서 춤추는 유니콘"
korean_prompt = "용과 함께 식사를"
korean_prompt = "사막에서 자라는 아이스크림 나무"
korean_prompt = "커피숍에서 일하는 로봇"
한국어 -> 영어 번역 ***

english_prompt = translator.translate(korean_prompt, src="ko", dest="en").text

print(f"한국어 프롬프트: {korean_prompt}")
print(f"번역된 프롬프트: {english_prompt}")
```

④ 이미지 생성

```
# 이미지 생성
num_images = 1 # 생성할 이미지 수
guidance_scale = 7.5 # 텍스트의 중요도 조정

# 이미지 생성 실행
images = pipeline(english_prompt, num_inference_steps=50, guidance_scale=guidance_scale, num_images_per_prompt=num_images).images

# 생성된 이미지 시각화
for idx, image in enumerate(images):
    plt.imshow(image)
    plt.axis("off")
    plt.title(f"Generated Image {idx+1}")
    plt.show()

# 이미지 저장 -> 이미지를 저장하고 싶으면 아래 코드 실행
for idx, image in enumerate(images):
    image.save(f"generated_image_{idx+1}.png")
    print(f"Image {idx+1} saved as 'generated_image_{idx+1}.png'")
```

