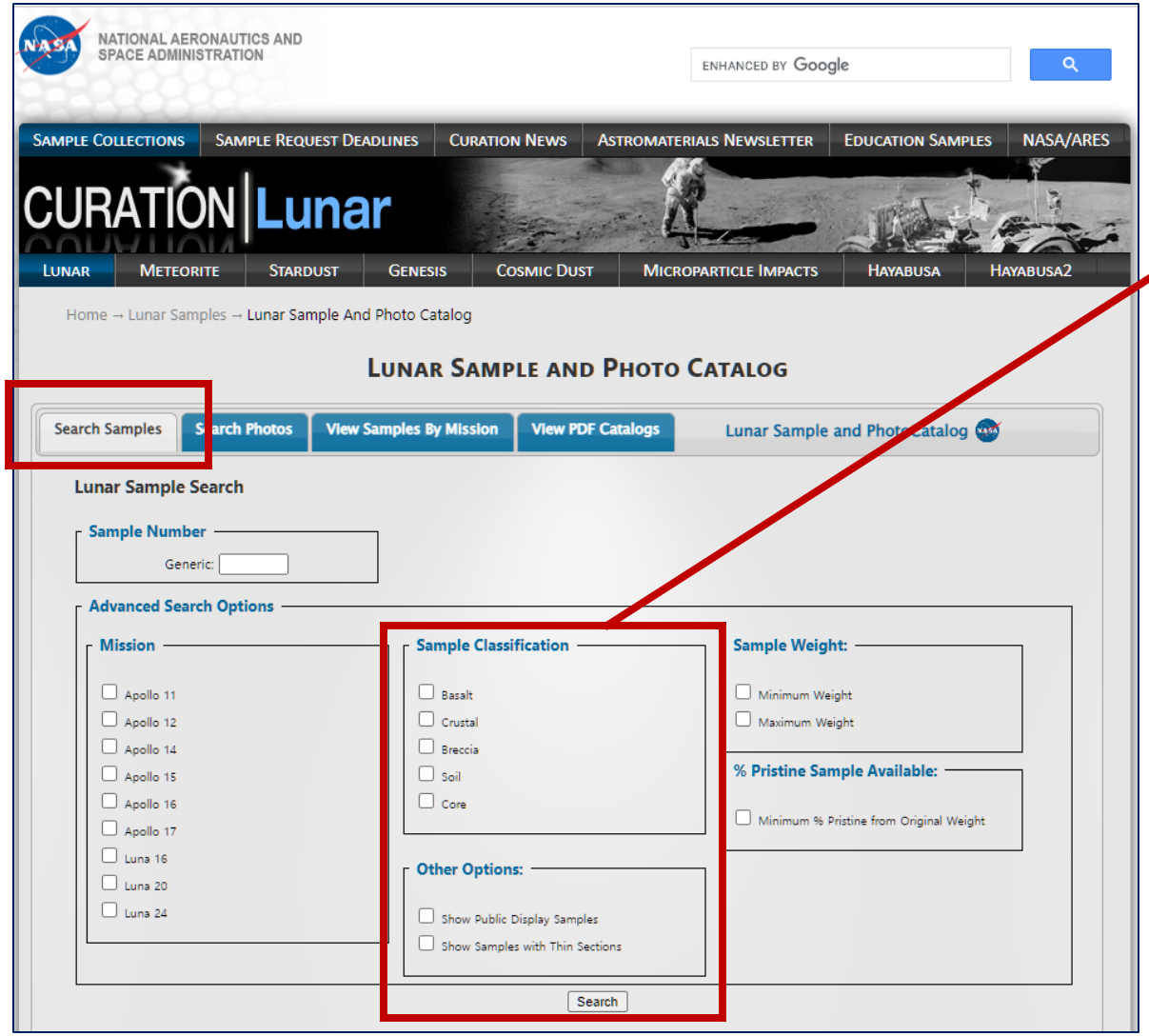
파이썬으로 배우는 딥러닝(Deep Learning)

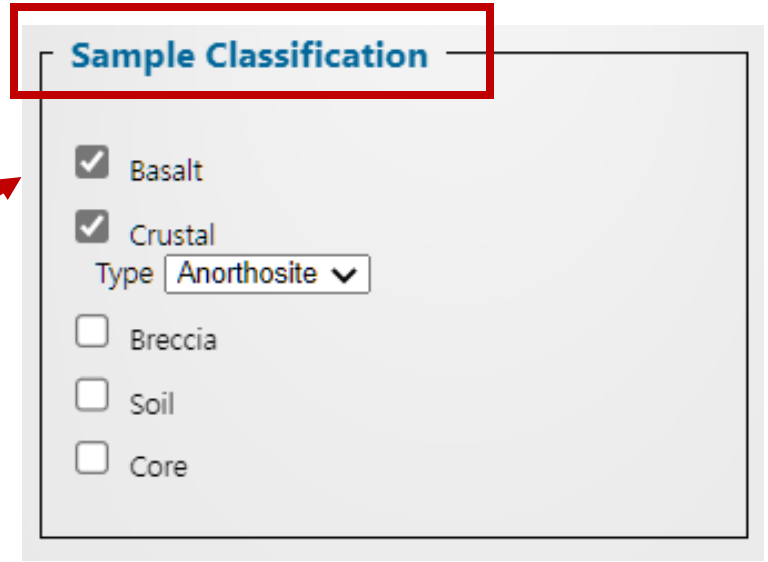
8회차 수업
전이학습 실습 : 암석식별머신만들기

목 차

NASA가 공개한 달 암석 이미지 수집

◆ 달 암석 이미지는 NASA의 Curation 페이지에서 찾을 수 있음.





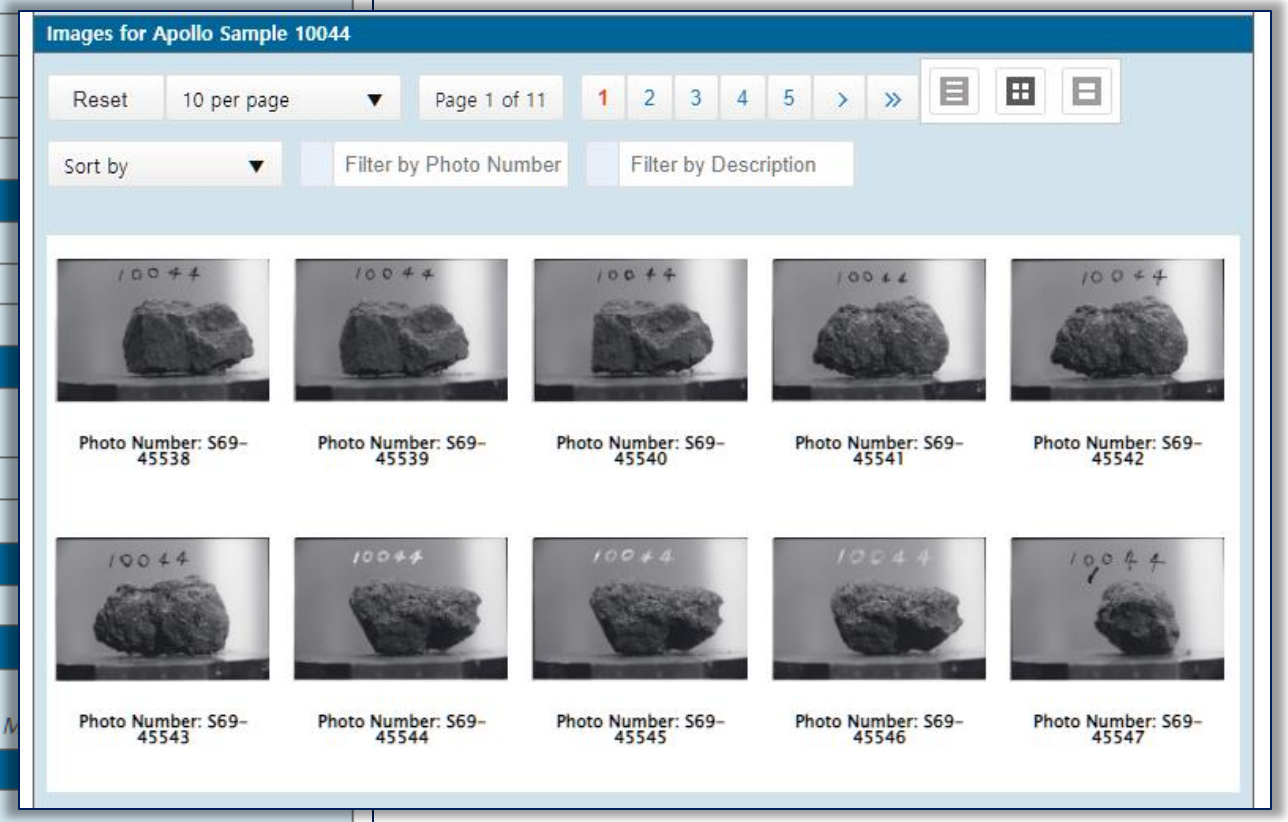
Basalt 와 Crustal(Anorthosite)를 선택하여 검색(search)

달 암석 이미지 확인 - 현무암

Apollo Sample 10044

Generic Number	10044
Mission Information	
Mission	Apollo 11
Station	
Landmark	
Bag Number	ALSRC 1003
Original Weight	247.50 g
Sample Classification	
Rock Type	Basalt
Rock Subtype	Cristobalite
Description	low K, medium-grained, ophitic, vugs, cristobalite
Sample Availability	
Percent of Pristine Sample Available	50.43%
Date of Pristinity Calculation	Dec 3, 2014
Thin Sections Available	See list of available thin sections.
Public Displays	
Display Samples Available	
Apollo Virtual Microscope @ the Virtual Microscope for Earth Sciences	
10044 – Ilmenite basalt	
The Apollo Virtual Microscope collection was developed in collaboration with The Open University, M	
Catalogs and Publications	
Apollo 11 Lunar Sample Information Catalog	
Catalog of Lunar Mare Basalts Greater Than 40 Grams: Part 1. Major and Trace Chemistry	
Lunar Sample Compendium	

아폴로 11호가 가져 온
현무암(Basalt) 사진

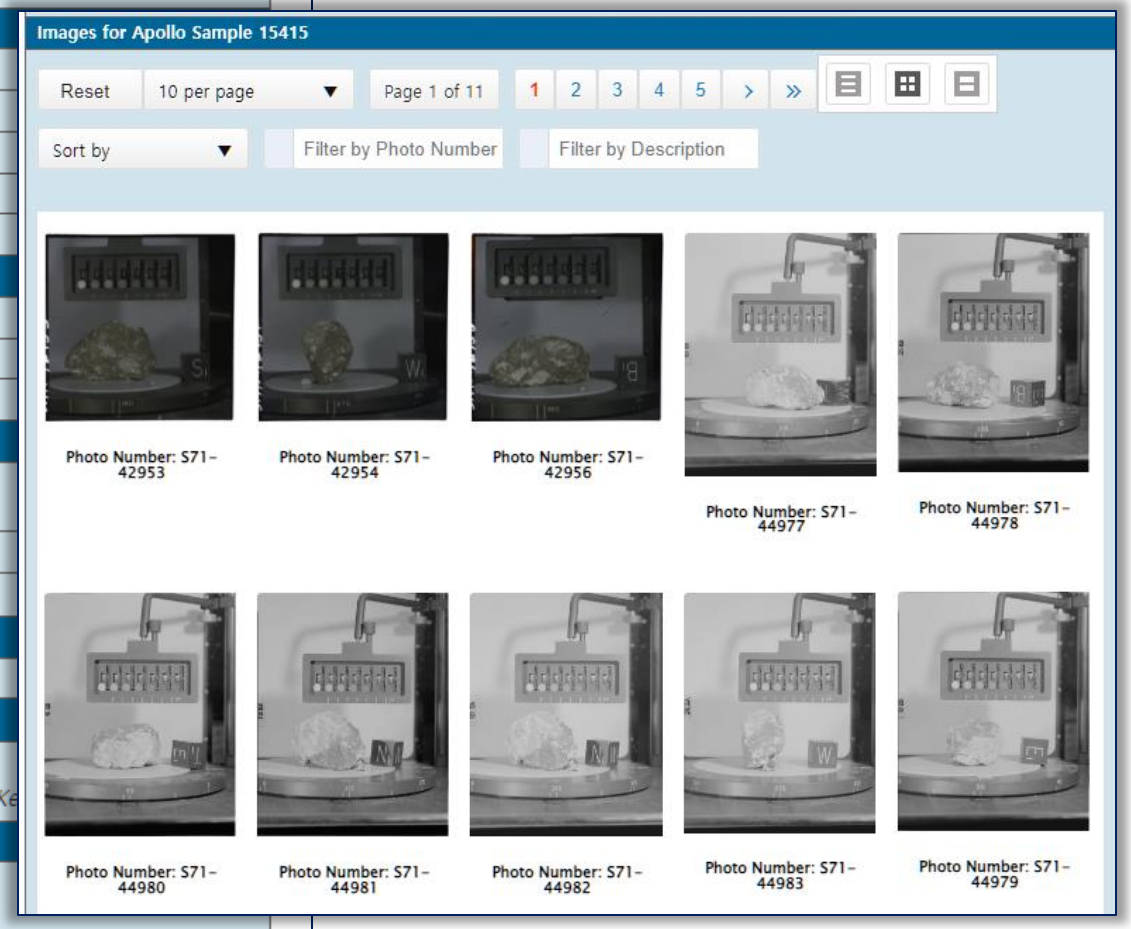


달 암석 이미지 확인 – 고지대 암석

Apollo Sample 15415

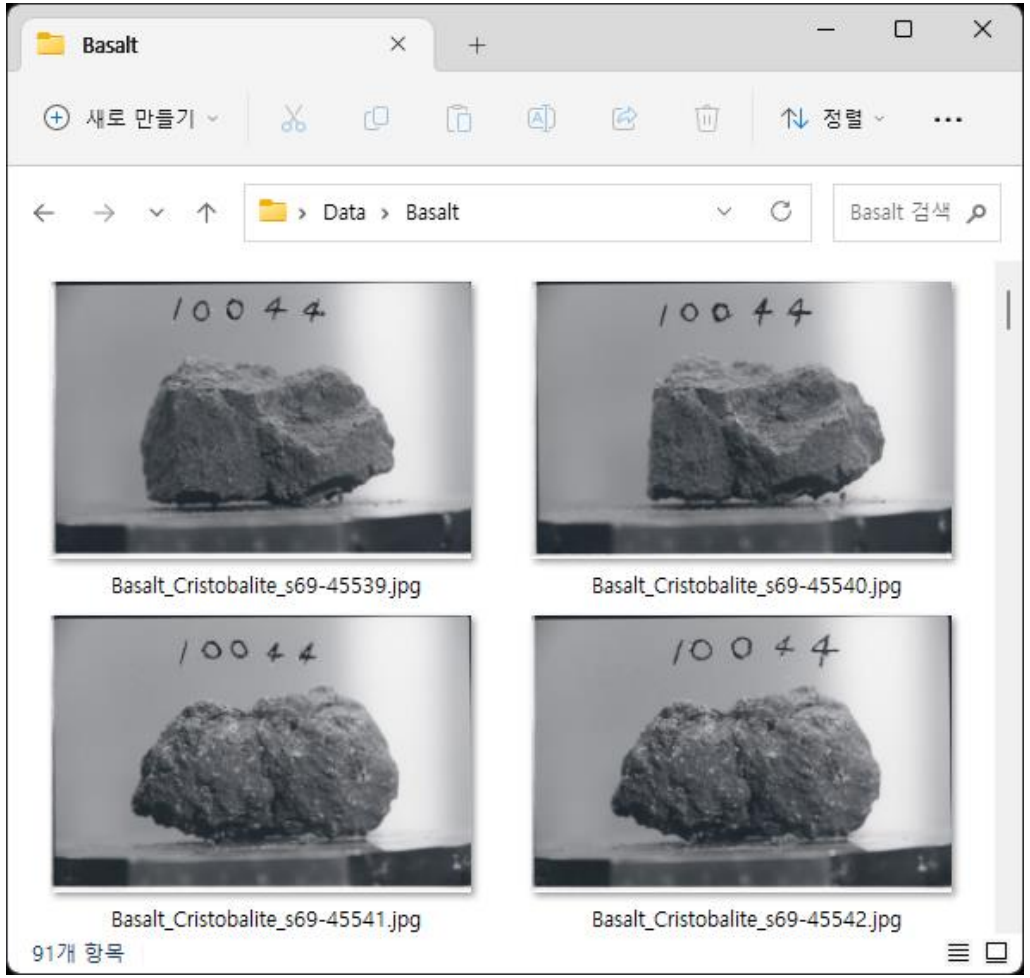
Generic Number	15415
Mission Information	
Mission	Apollo 15
Station	7
Landmark	SPUR CRATER
Bag Number	SCB 3 196
Original Weight	269.40 g
Sample Classification	
Rock Type	Crustal
Rock Subtype	Anorthosite
Description	Ferroan anorthosite
Sample Availability	
Percent of Pristine Sample Available	67.07%
Date of Pristinity Calculation	Mar 2, 2015
Thin Sections Available	See list of available thin sections.
Public Displays	
Display Samples Available	
Apollo Virtual Microscope @ the Virtual Microscope for Earth Sciences	
15415 (16) Ferroan Anorthosite	
The Apollo Virtual Microscope collection was developed in collaboration with The Open University, Milton Ke	
Catalogs and Publications	
Lunar Sample Information Catalog	
Lunar Sample Compendium	
Catalog of Apollo 15 Rocks: Part 2. 15306-15468	

아폴로 15호가 가져 온
고지대 암석(Crustal) 사진

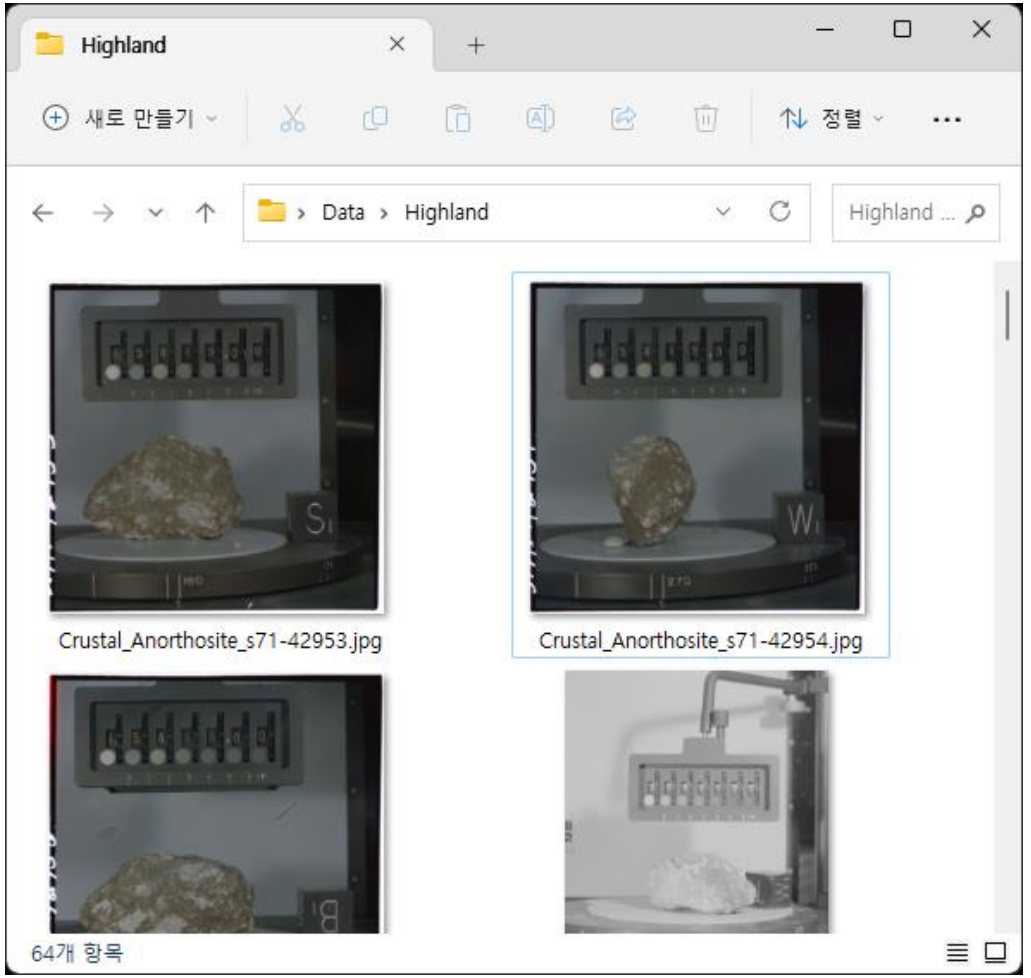


실습 데이터(암석 이미지 데이터 등) 다운로드/저장

Data/Basalt 디렉토리 확인



Data/Highland 디렉토리 확인

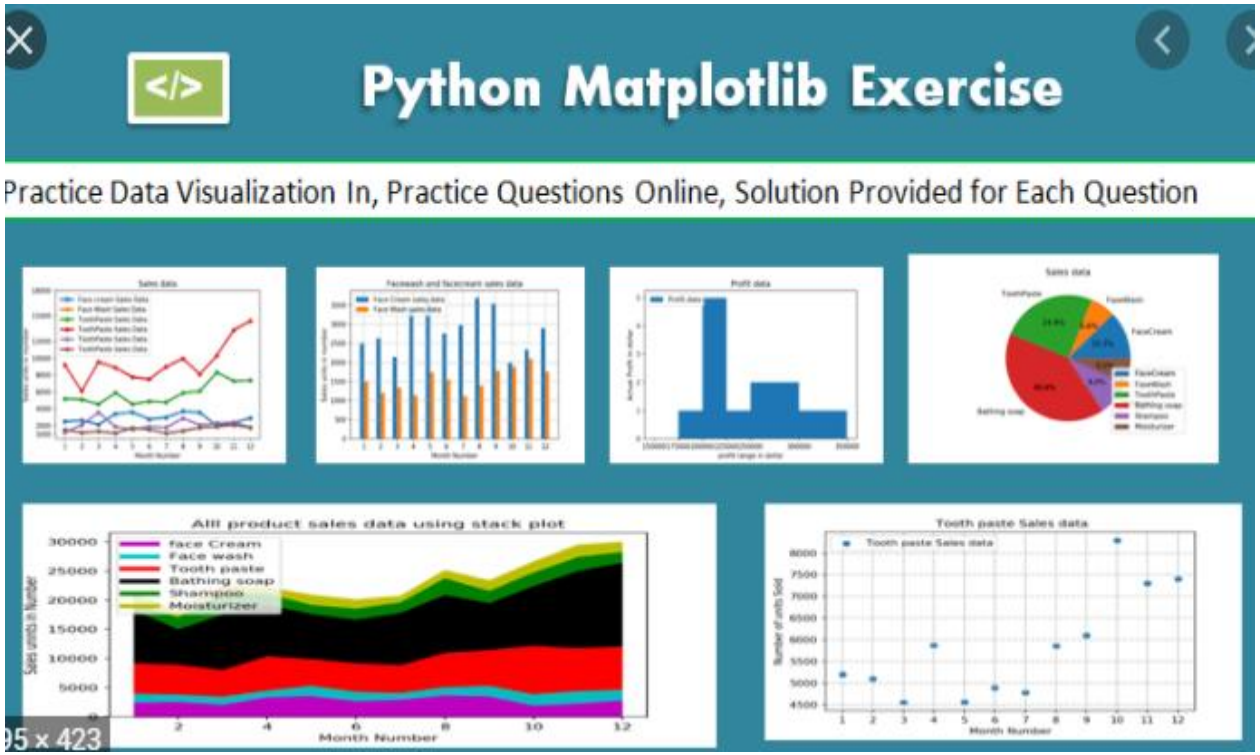


목 차

AI 프로젝트를 위한 Python 라이브러리

◆ Matplotlib

- 데이터의 시각화에 사용되는 패키지. 다양한 저수준 API 제공
- PyPlot : matplotlib의 서브 패키지. 간단한 시각화를 위한 API 제공



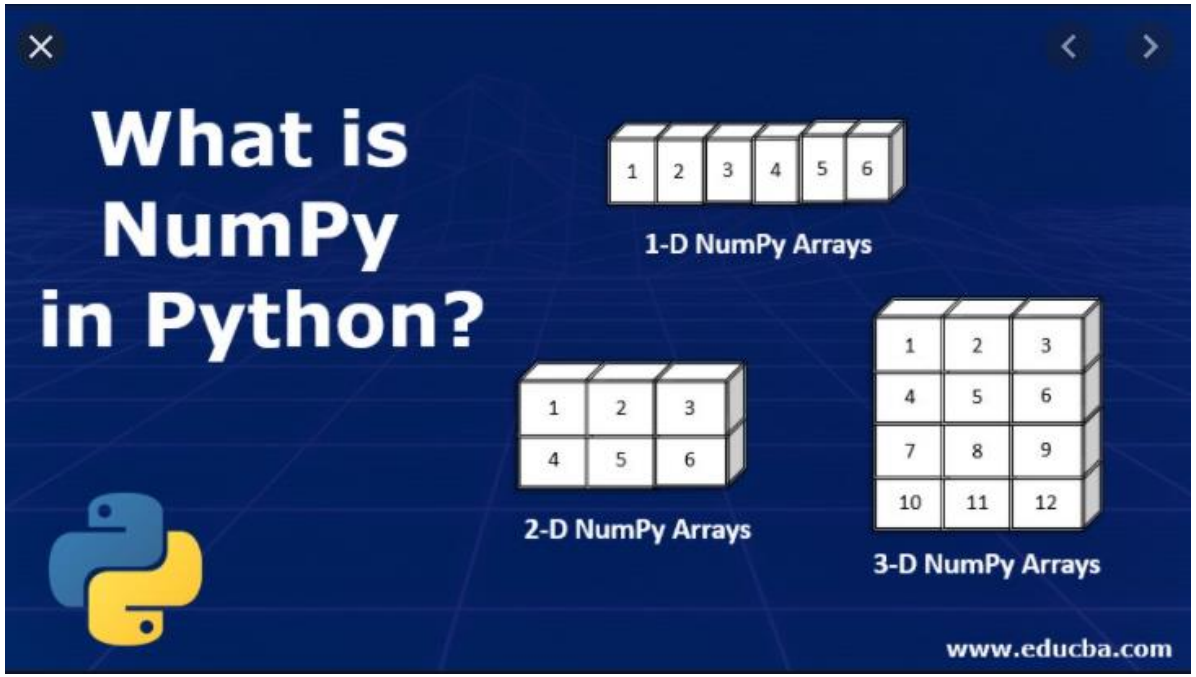
- **import** matplotlib.pyplot **as** plt :
  matplotlib.pyplot 모듈을 plt라는 이름으로 불러들인다.
- **%matplotlib inline** :
  그래프를 jupyter notebook 셀에서 보여준다.
- **plt.figure(figsize=(20,15))** :
  그래프를 그릴 화면 사이즈를 정한다(가로 20, 세로 15)
- **plt.axis('off')** :
  그래프를 그릴때 축을 표시하지 않도록 설정한다.
- **plt.imshow(image)** :
  RGB 데이터 또는 2D array 구성된 이미지를 표시한다.
- **plt.show()** :
  화면에 그래프를 나타낸다.

https://pynative.com/python-matplotlib-exercise/

AI 프로젝트를 위한 Python 라이브러리

◆ **NumPy**

- Numerical Python을 줄여서 표기한 것
- 수학 및 과학 연산을 위한 패키지
- 수치해석 및 통계 분야에 널리 사용됨



- NumPy에는 배열 및 행렬 데이터를 조작을 위한 유용한 함수들을 제공한다.

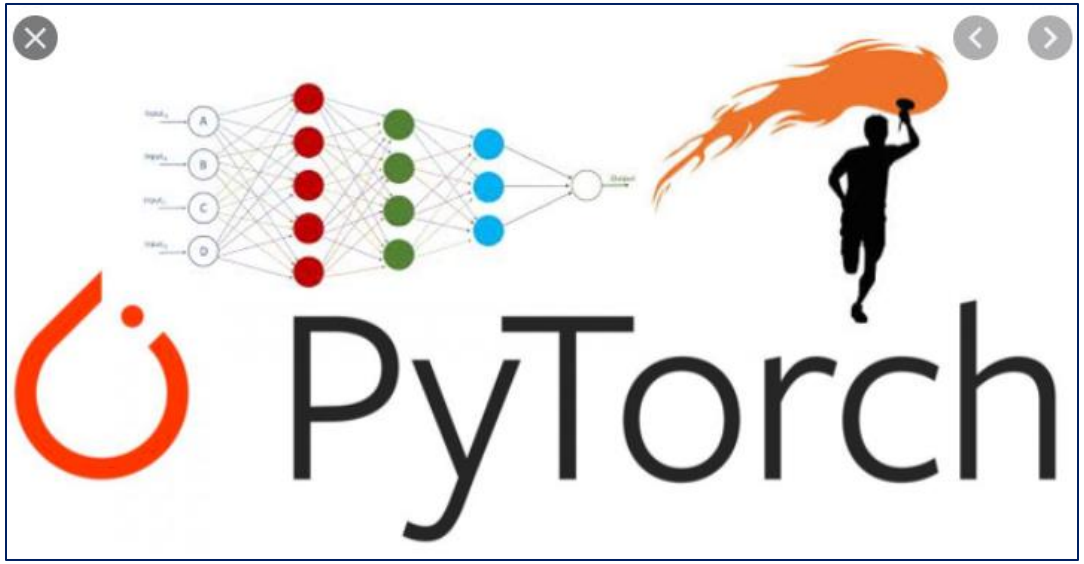
- **import** numpy **as** np :
  numpy 라이브러리를 np라는 이름으로 불러들인다.
- **np.floor(float_num)** :
  실수 파라미터 float_num을 버림한다(바닥함수).
- **np.random.shuffle(indices)** :
  배열 파라미터 indices를 랜덤하게 섞는다.

https://www.educba.com/what-is-numpy-in-python/

AI 프로젝트를 위한 Python 라이브러리

◆ PyTorch

- AI 라이브러리로서 신경망을 만들기 위한 함수들 제공



- **import** torch : torch 라이브러리를 불러들임
- **from** torch **import** nn : torch.nn 모듈은 신경망을 구성하는데 필요한 모든 구성 요소 제공
- **from** torch **import** optim : torch.optim 모듈. 다양한 최적화 알고리즘 제공
- **from** torch.autograd **import** Variable : torch.autograd 모듈은 텐서의 모든 연산에 대해 자동 미분을 제공. torch.autograd.Variable 클래스는 텐서 기반으로 정의된 연산을 지원하고 계산이 완료된 후 .backword()를 호출하여 모든 gradient를 자동으로 계산
- **import** torch.nn.functional **as** F : 컨벌루션 함수, 풀링 함수, 활성화 함수, 선형 함수, Dropout 함수, 손실 함수 제공. 이 함수들을 사용하여 각 layer 구성 가능

https://ichi.pro/ko/pytorchleul-sayonghan-dib-leoning-eun-gomun-i-anibnida-196777976894010

AI 프로젝트를 위한 Python 라이브러리

◆ Torchvision

- Pytorch와 함께 사용되는 **computer vision**용 라이브러리
- 효율적인 이미지 및 비디오 변환을 위한 유틸리티와 사전 학습된 모델 및 데이터 세트 제공
  (별도로 설치해서 사용해 함)

- **import** torchvision : torchvision 라이브러리를 불러들임
- **from** torchvision **import** datasets :
  torchvision.datasets는 가장 일반적으로 사용되는 데이터셋을 쉽게 다운로드 할 수 있는 다양한 옵션 제공
- **from** torchvision **import** transforms :
  torchvision.transforms는 이미지 확대/변환시에 사용되는 모듈
- **from** torchvision **import** models :
  torchvision.models는 전이 학습을 용이하게 하기 위해 이미지 분류, 객체 감지 등에 대한 사전 학습된 모델이 있음. 필요한 모델을 로컬 시스템에 다운로드하여 전이학습 또는 fine-tuning하여 사용

https://learnopencv.com/pytorch-for-beginners-semantic-segmentation-using-torchvision/

GitHub 소개

- git[1] 의 repository를 서비스로 제공하기 시작하였고 (2007년), 수많은 오픈소스 프로젝트들이 GitHub에 모여들면서 현재는 압도적으로 많은 사용자들이 이용하고 있음

1) Git이란?
   - 분산형 소프트웨어 형상 관리 도구 (Distributed Software Configuration Management)
   - 과거 RCS,CVS, SVN 등 다양한 도구가 있었으나 Git의 분산형 관리 모델이 등장하면서 현재는 Git이 업계를 대표하는 도구가 됨

- 2018년 마이크로소프트에 인수된 이후 오픈소스 생태계에 대한 지원이 줄어들 것이라는 우려와는 달리 마이크로소프트와의 시너지 효과를 통해 그 영역을 확대하고 있음
- **단순 코드 호스팅 플랫폼이 아닌 개발 플랫폼으로 확장** 중
- Repository 서비스 이외에도 Collaborative Coding, Automation & CI/CD, Security, Client Apps, Project Management, Team Administration, Community 등 소프트웨어 개발 관련 다양한 분야에 서비스 제공
- GitHub의 이름에서 나타내는 것과 같이 개발환경의 'Hub'역할을 가능하게 하는 다양한 서비스와의 연계 용이함
- **Codespaces 서비스를 이용하여 Python 개발이 가능하며 또한 Web 지원으로 언제 어디서나 개발이 가능함**

Python 개발 환경 비교

	MS VS Code	Jupyter Notebook	Google Colab	Azure Notebook	GitHub Codespaces
공급사	Microsoft	Project Jupyter	Google	Microsoft	GitHub (Microsoft 자회사)
유형	Desktop	Desktop with browser	SaaS	SaaS	SaaS
필요조건	Python3 Visual Studio Code Extension	Anaconda(with Python3)	Browser Google 계정	Browser only Azure 계정	Browser GitHub 계정
가격	• 무료	• 무료	• 무료 • Pro $9.99/월 • Pro+ $49.99/월	• 유료 (Pay as you go)	• 무료 • Pro/Team $4/월 • Enterprise $21/월
장점	• 비용 발생 없음 • Visual Studio Code의 다양한 기능 활용 가능 • GitHub 연동 가능	• 비용 발생 없음 • GitHub 연동 가능	• 사용량이 많지 않은 경우 무료로 사용 가능 • Google 기반 workflow 구성이 비교적 용이	• 쓰는 만큼 비용 지불 • Azure cloud내 다른 작업과 연동 용이	• 사용량이 많지 않은 경우 무료로 사용 가능 • GitHub 기반 workflow 구성 용이 (CI/CD 등)
단점	• PC에 소프트웨어 설치 및 관리 노력 필요	• PC에 소프트웨어 설치 및 관리 노력 필요	• 무료 버전으로 충분하지 않을 수 있음	• 무료 버전 없음 • Compute에 따라 비용이 많이 발생할 수 있음 • Azure cloud를 사용해본 경험이 필요	• 무료버전으로 충분하지 않을 수 있음
GitHub 연동	가능	가능	가능	가능	용이
성능	PC 성능에 비례	PC 성능에 비례	• 무료의 경우 제한적 • 유료는 보다 많은 리소스 제공	선택하는 Compute 성능에 비례	• 무료의 경우 제한적 • 유료는 보다 많은 리소스 제공
특이사항				Azure cloud college students 무료 계정 제공 활용	학생/교사에 대해서 할인 제공
권장	고성능 PC 보유 VS code 등 PC 사용 능숙	Anaconda를 이미 설치	Google기반 환경의 사용자	Azure cloud를 이미 사용 중인 대학생	학생 및 교사

- 가정에서: (특히 PC에 고성능 그래픽 카드 (예: RTX3070)가 장착되어 있고) PC 활용이 능숙한 경우
- 학생/교사 등 교육 현장에서 무료 이용 가능

GitHub Codespaces에서 Python 개발

계정 생성

- https://github.com에서 계정 생성
- Education program이 존재해서 GitHub에서 제공하는 다양한 서비스를 큰 비용 부담 없이 사용가능 (GitHub Pro 무료 제공)

Codespaces 시작

- Visual Studio Code for the web이 GitHub에 연동
- 최근 소프트웨어 개발 시 이미 개발되어 있는 다양한 라이브러리 등을 활용하는 것이 필수적임
- 개발 초기에 이러한 요소를 구성하는 것이 생각보다 쉽지 않음
- Codespaces에서 제공하는 template 사용 (Python의 경우 Jupyter template이 제공됨)

Repository 생성

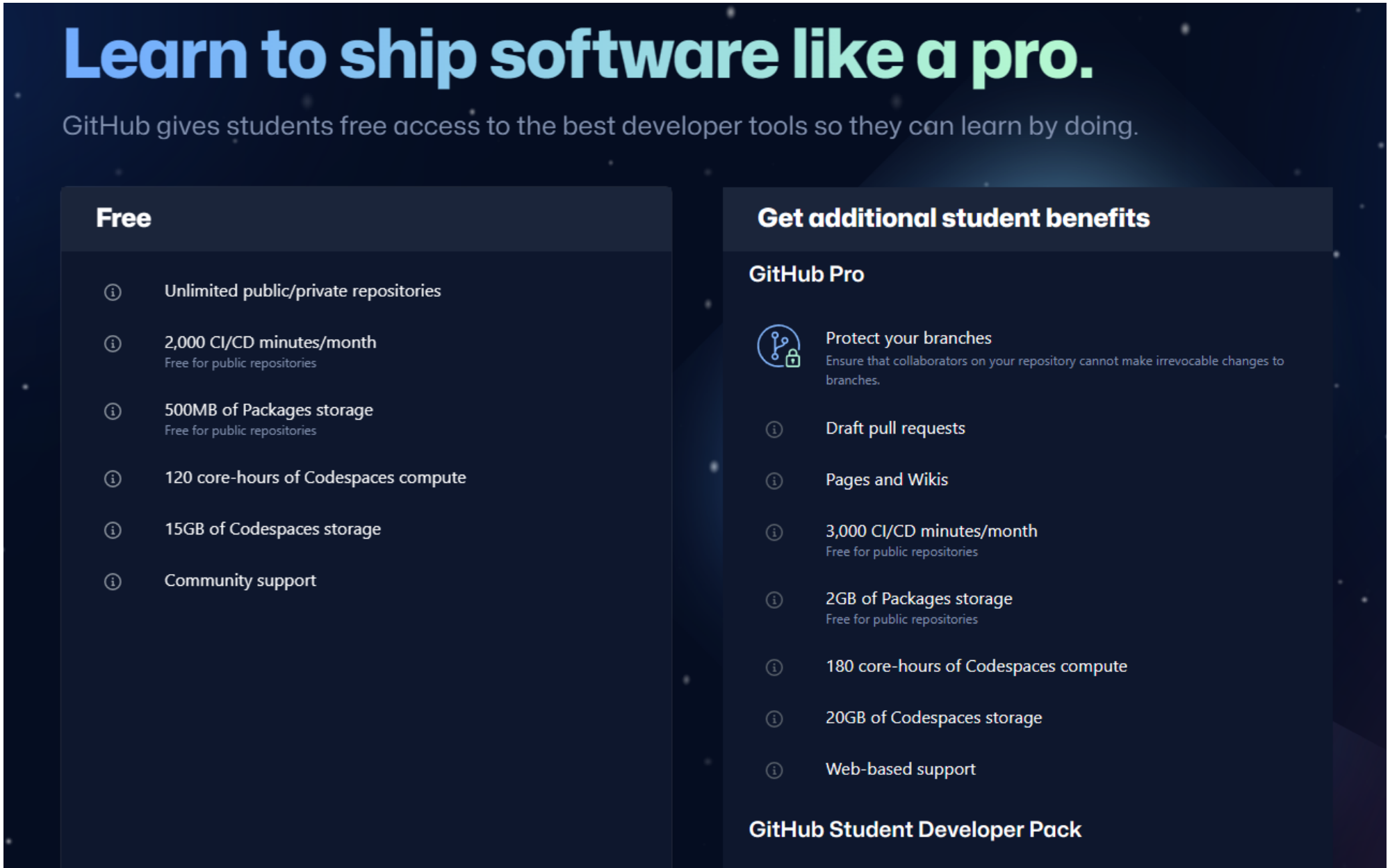
- Repository란 말 그대로 코드를 저장하는 저장소
- 새로운 프로젝트를 시작하기에 앞서 우선 repository를 만들고 관련된 모든 내용들을 같이 저장하면 관리가 용이함.
- 실습 시 편의상 Codespaces에서 template을 사용해서 만들어진 리소스를 repository에 저장하는 방식 사용

Codespaces 중지 및 재시동

- Codespaces가 실행되고 있는 동안(python code가 실행되지 않고 편집상태인 경우도 포함)에도 과금됨
- 사용이 끝난 경우 일시 정지 시키면 비용 절약 가능

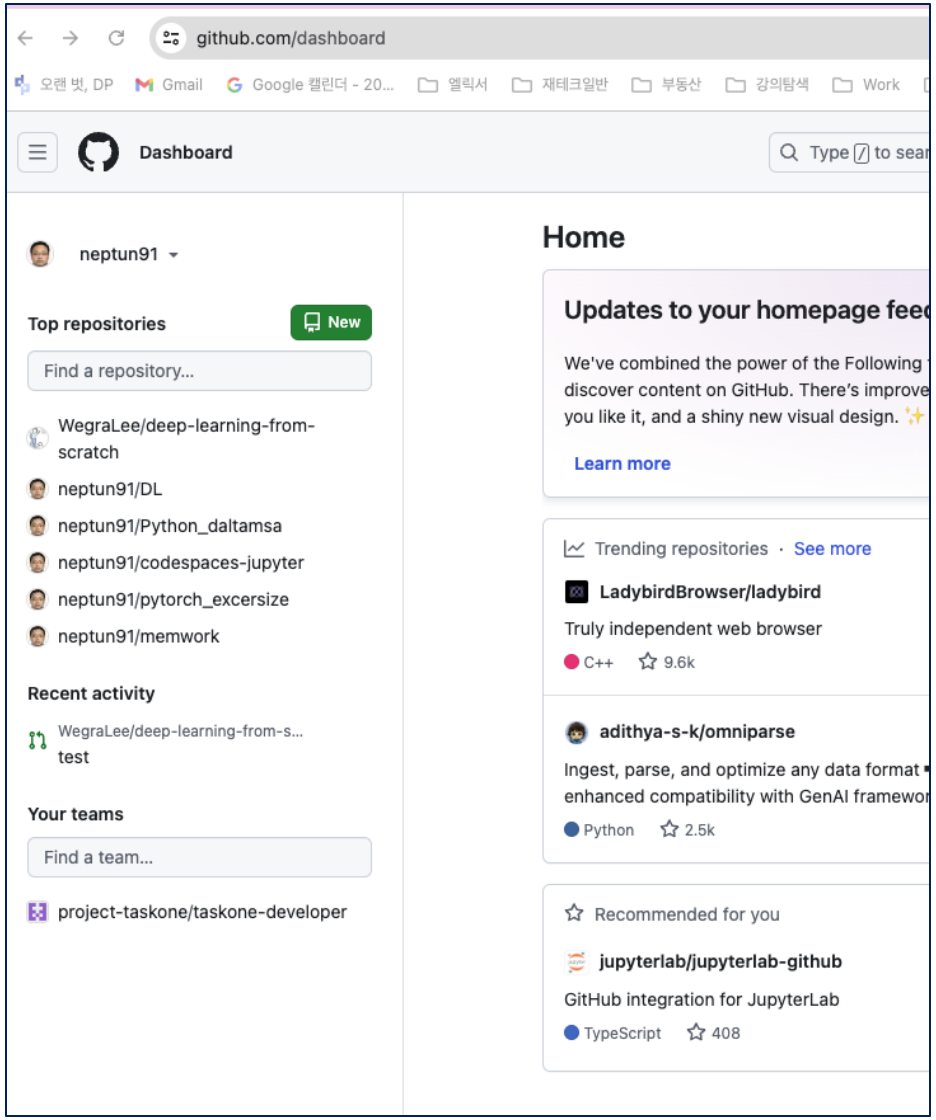
일반적인 workflow는 repository를 먼저 구성하지만, Codespaces의 template을 활용하기 위해 위와 같은 순서로 구성함

GitHub Free vs Pro

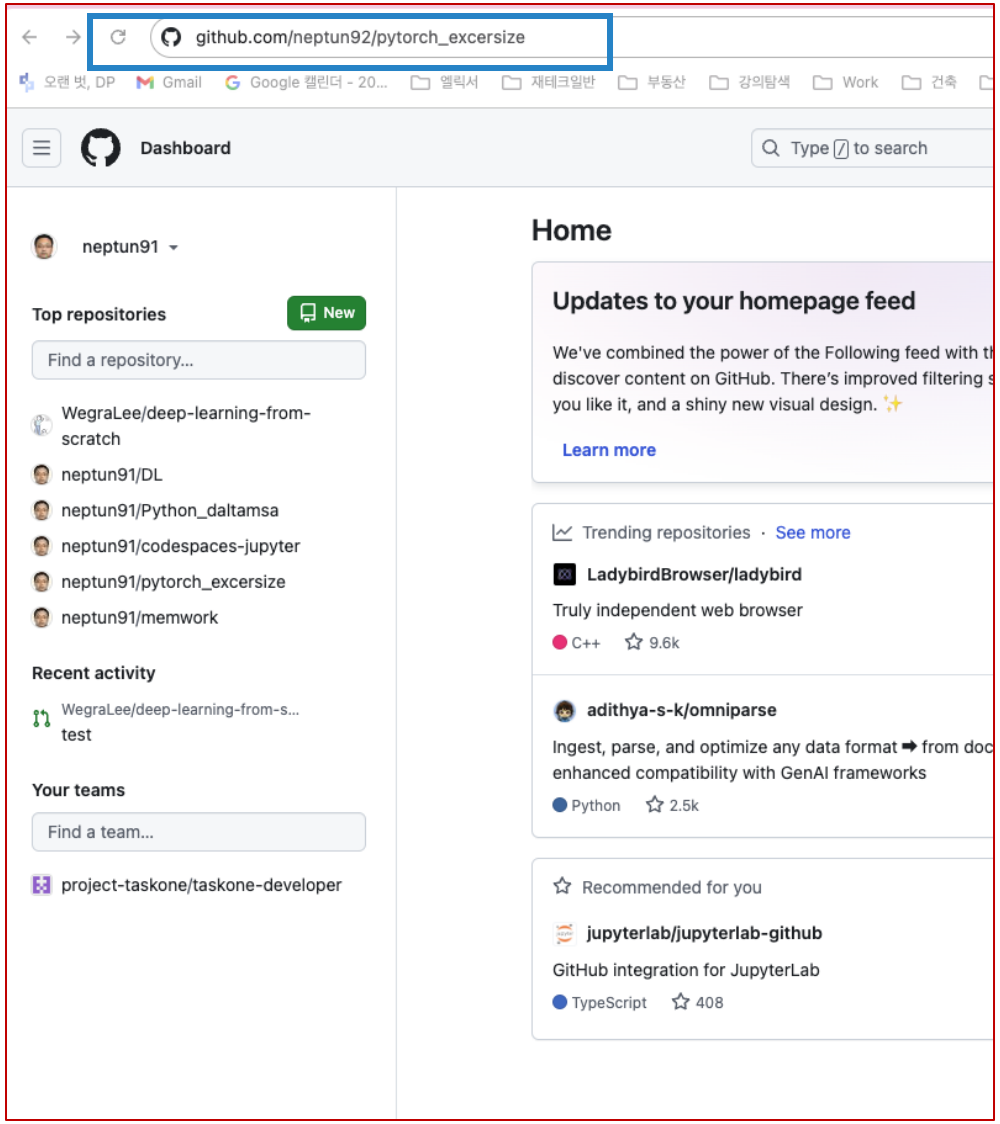


GitHub codespace 생성(1)

Step1. GitHub 로그인 상태

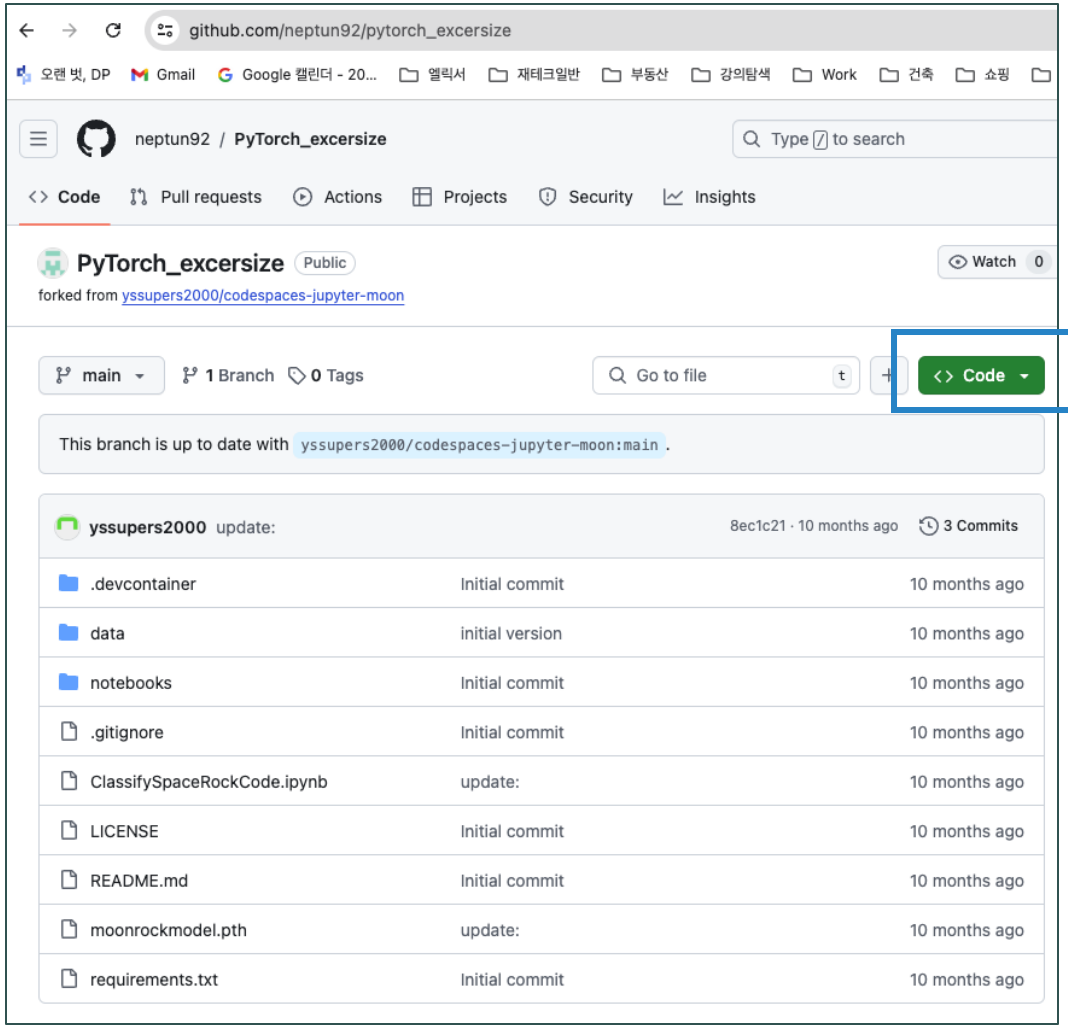


Step2 주소창입력 : https://github.com/neptun92/DL-Excersize

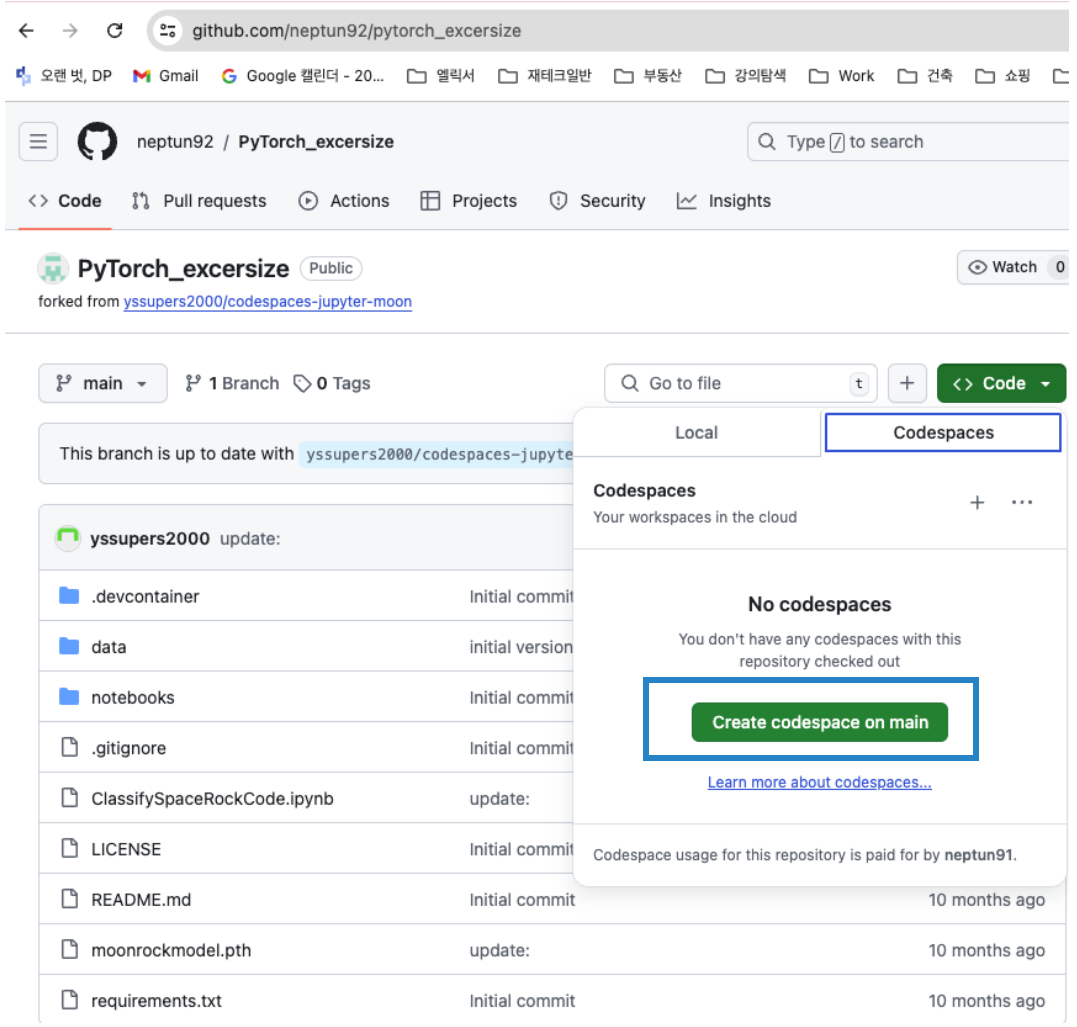


GitHub codespace 생성(2)

Step3. pytorch_excersize 소스 화면 → "<> Code" 버튼 클릭



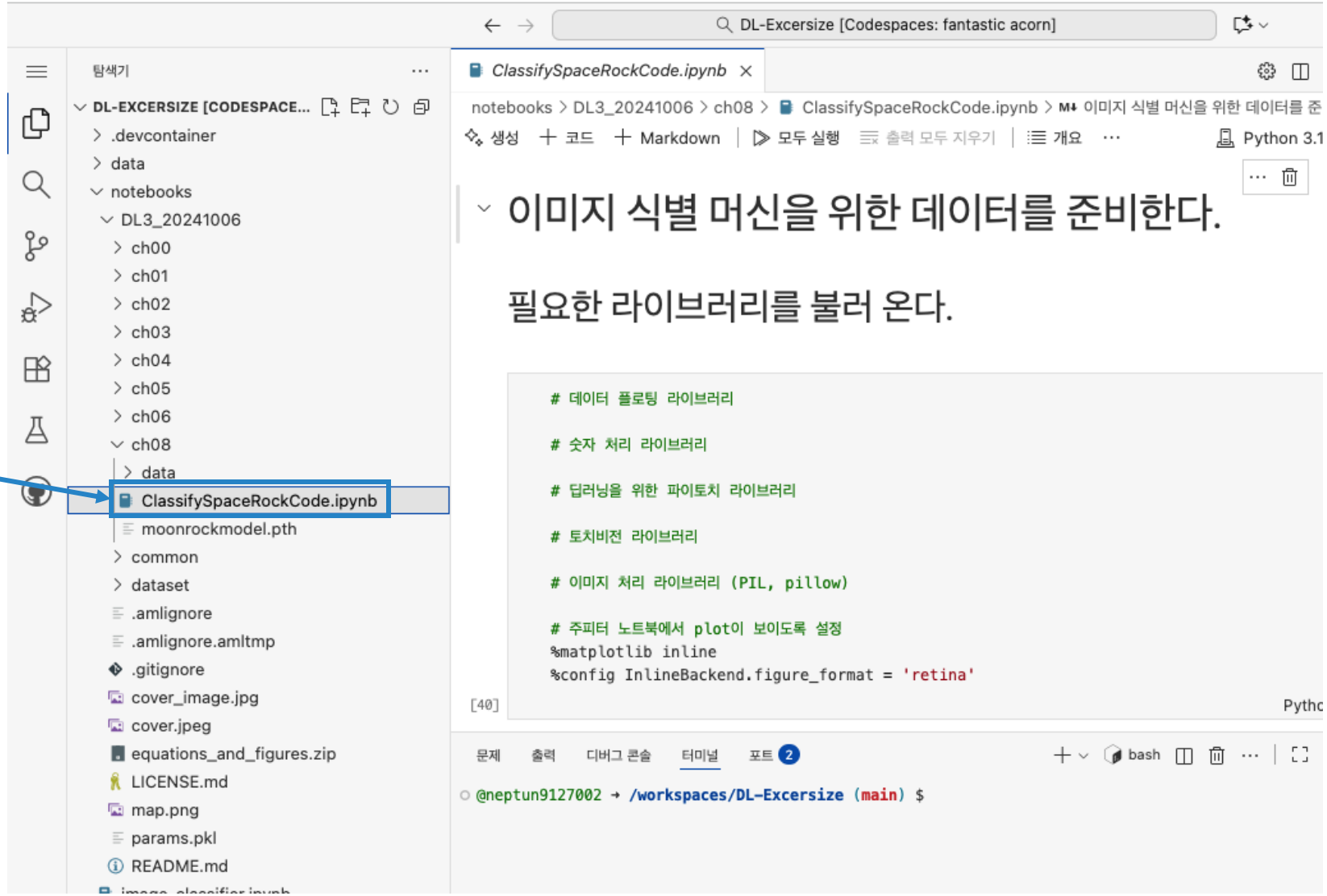
Step4 'Create codespace on main' 클릭



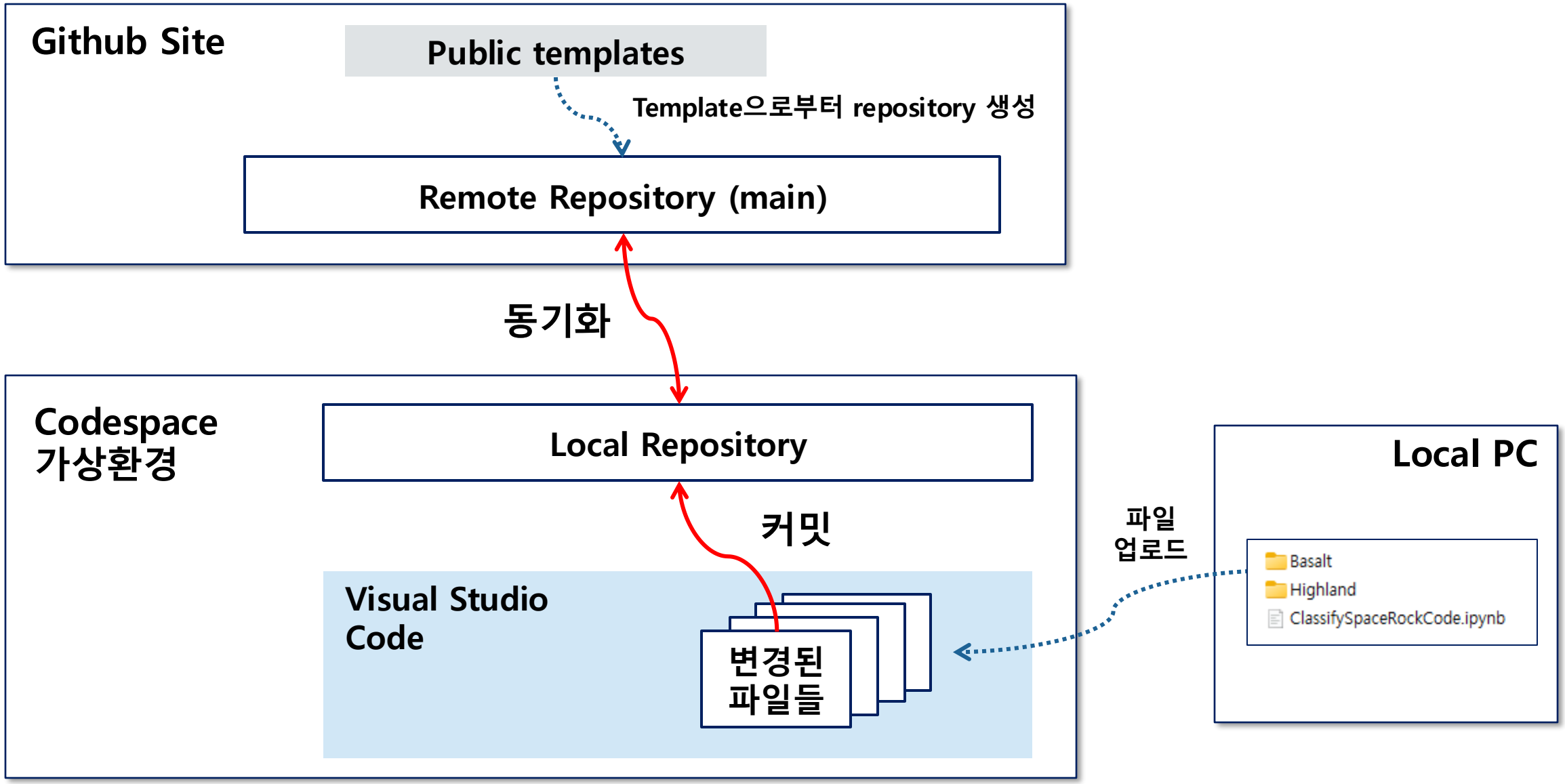
GitHub codespace 생성(3)

Step5. codespace 생성 완료

실습파일

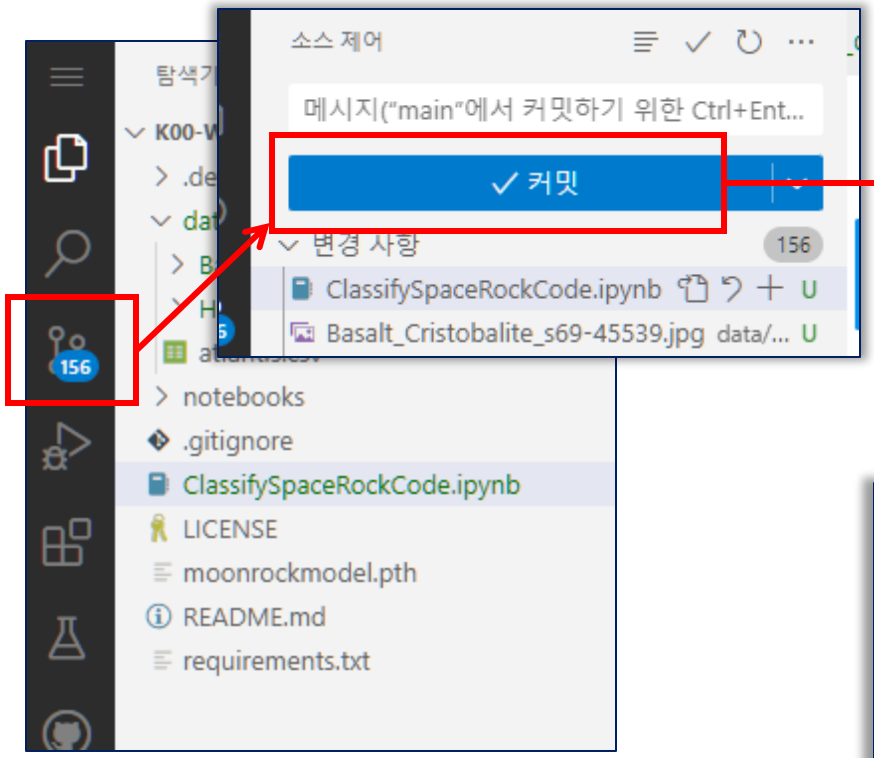


업로드 파일 커밋 및 동기화 - 개요

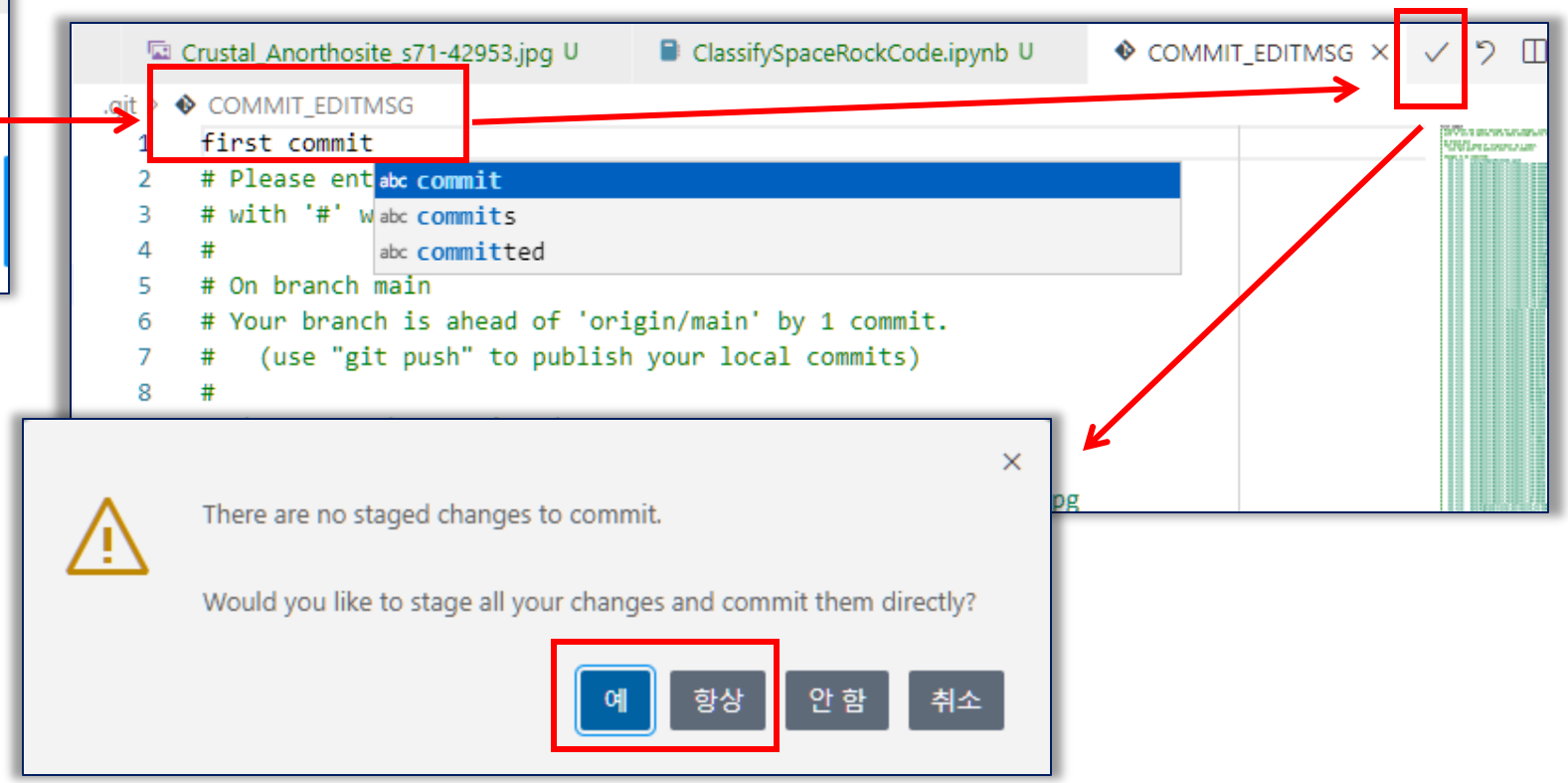


업로드 파일 커밋 및 동기화 - 커밋

소스제어 창에서 '커밋' 클릭



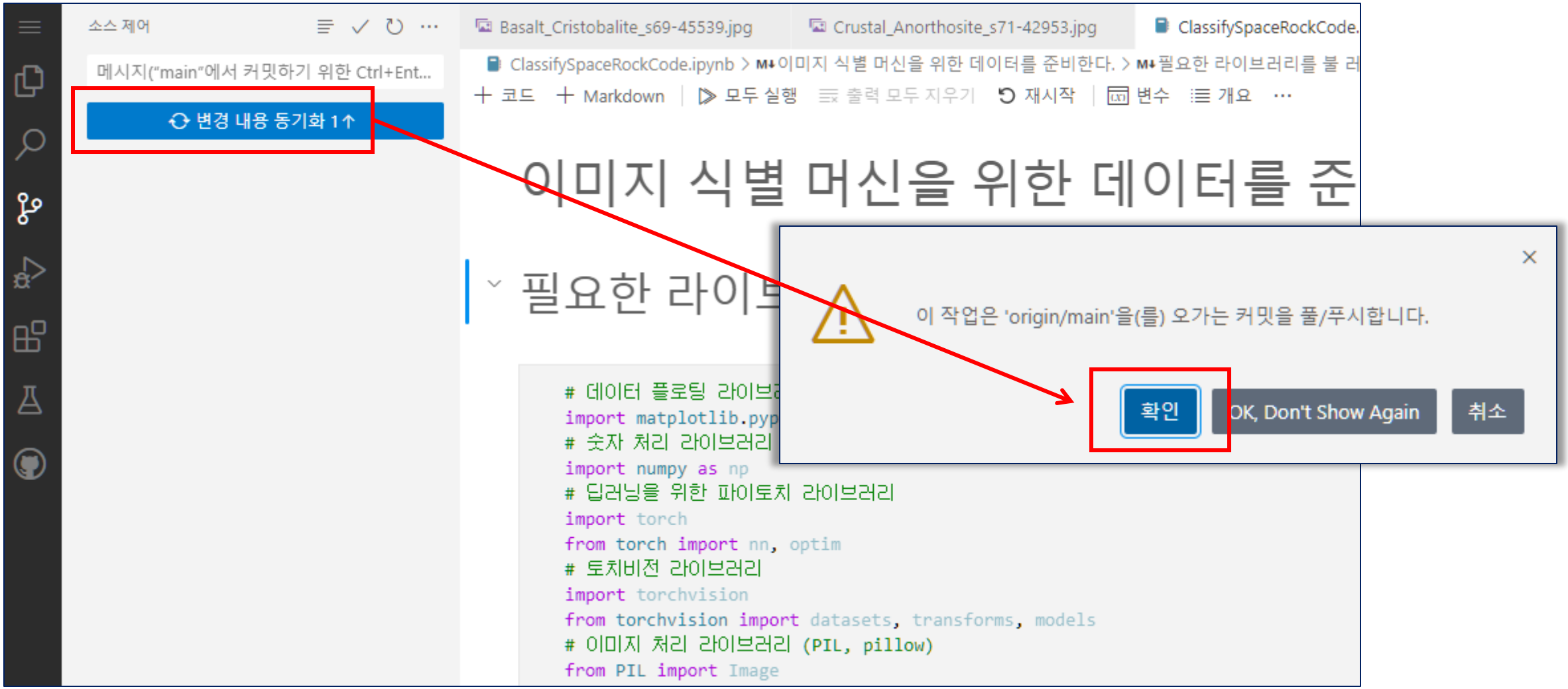
커밋 메시지 입력 후
체크버튼 클릭



'예' 또는 '항상' 클릭

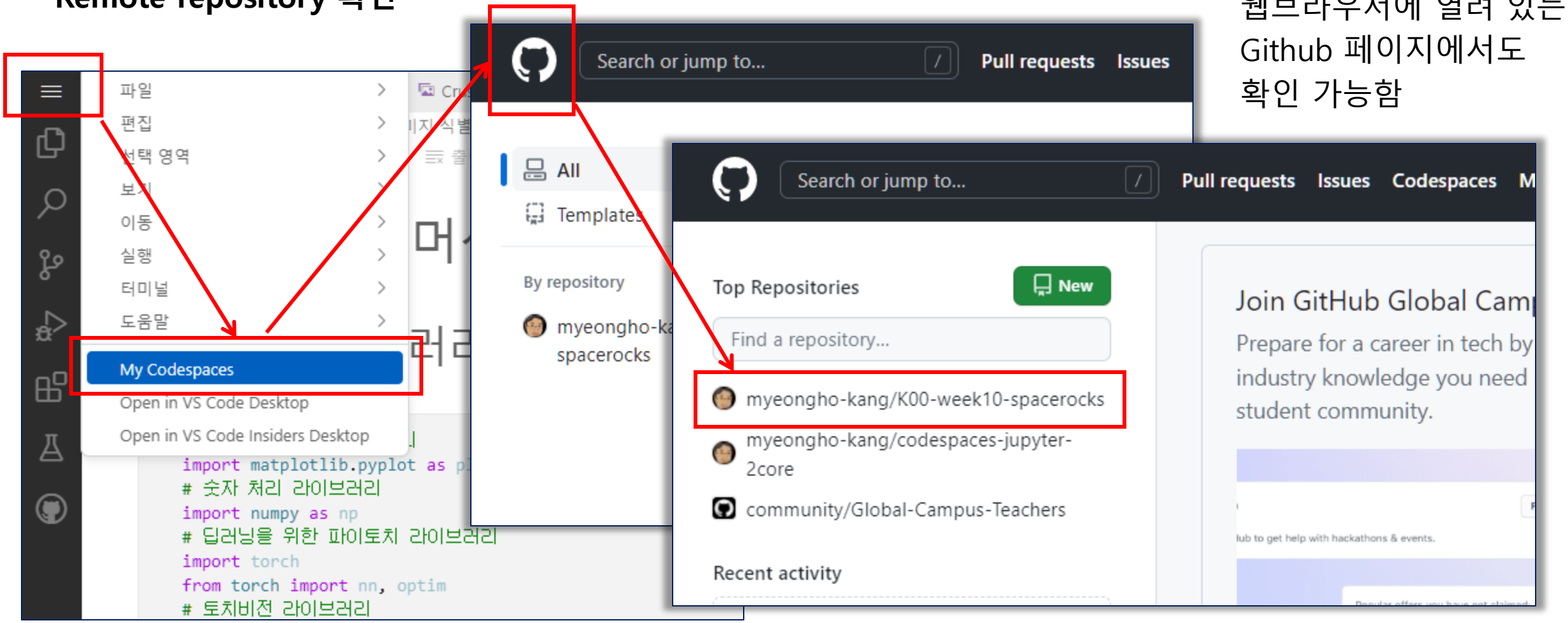
업로드 파일 커밋 및 동기화 - 동기화

변경 내용 동기화를 통해 remote repository에 반영



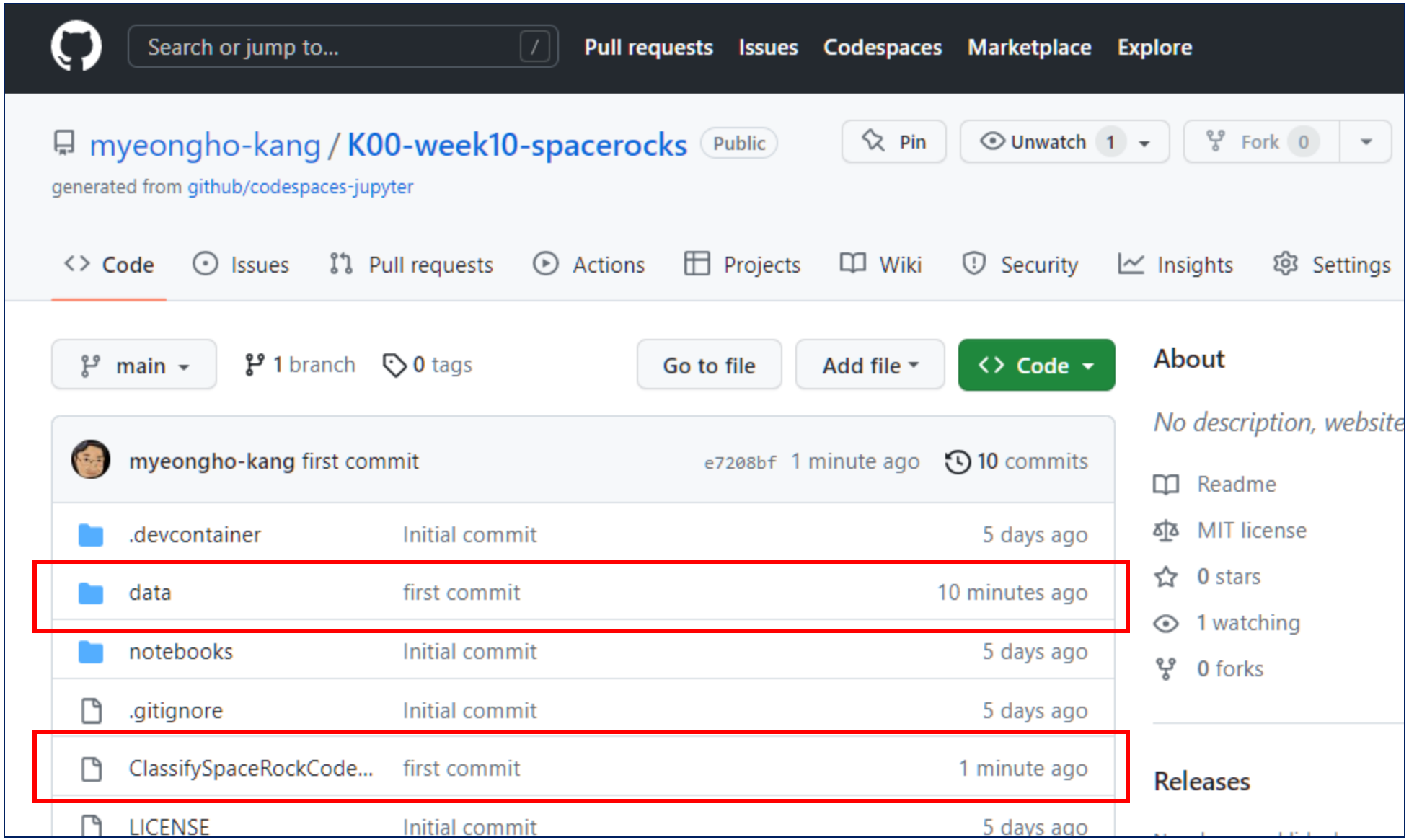
업로드 파일 커밋 및 동기화 – Remote repository 확인

My Codespaces 페이지 → Github 초기화면 → repository 주소를 클릭하여 Remote repository 확인



웹브라우저에 열려 있는 Github 페이지에서도 확인 가능함

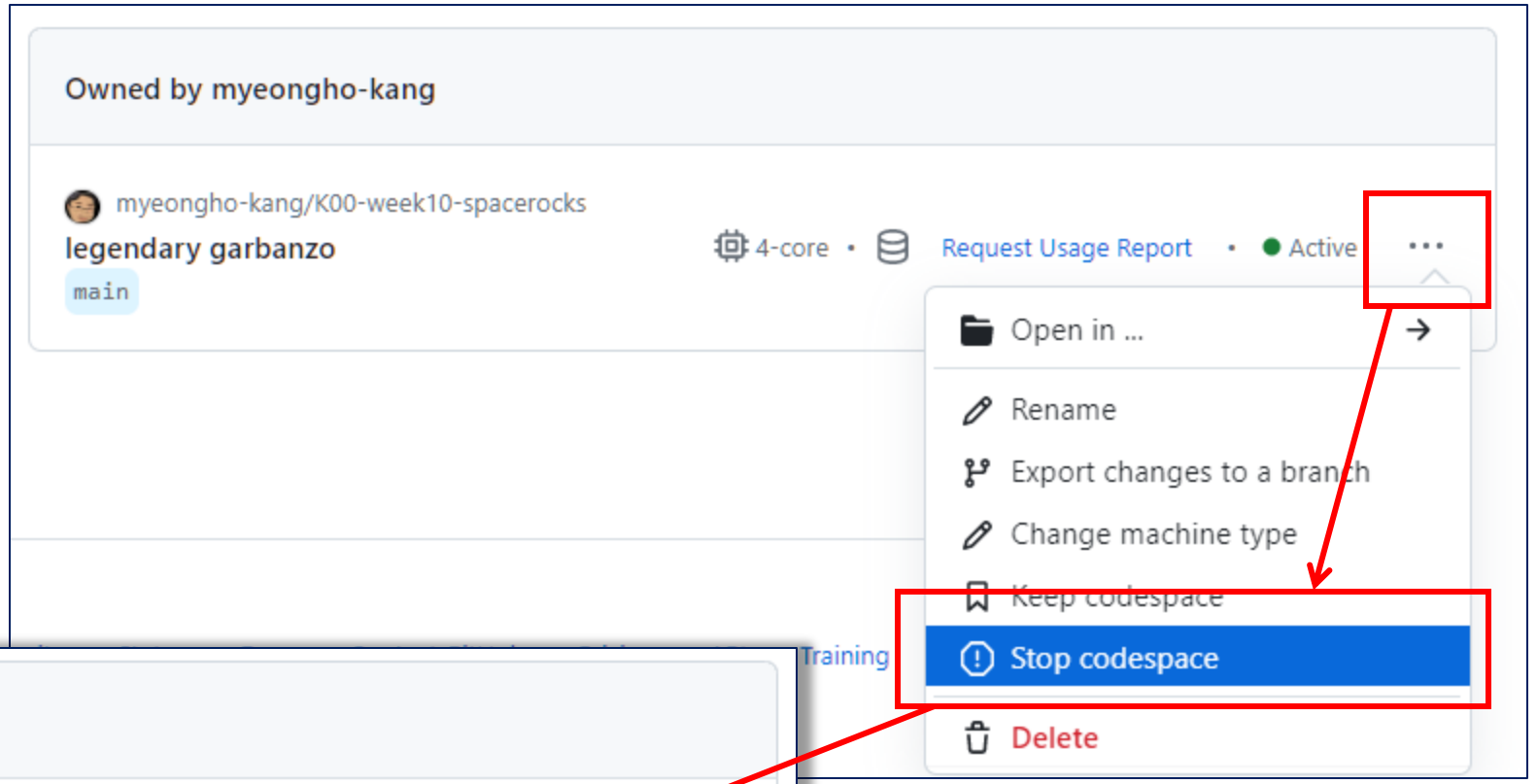
업로드 파일 커밋 및 동기화 – Remote repository 확인

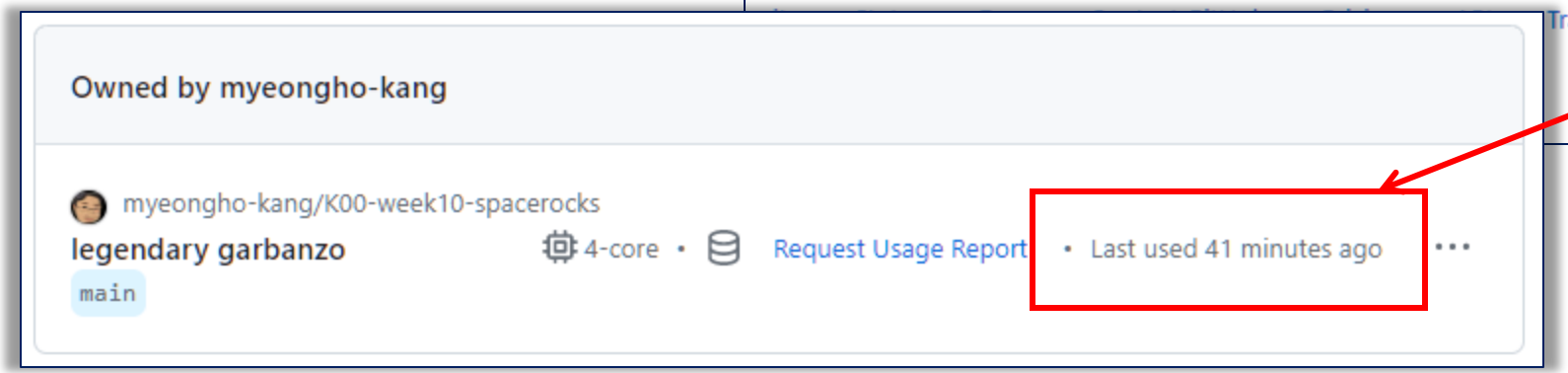


Remote repository에 반영된 내용 확인

참고 : Codespace의 중지

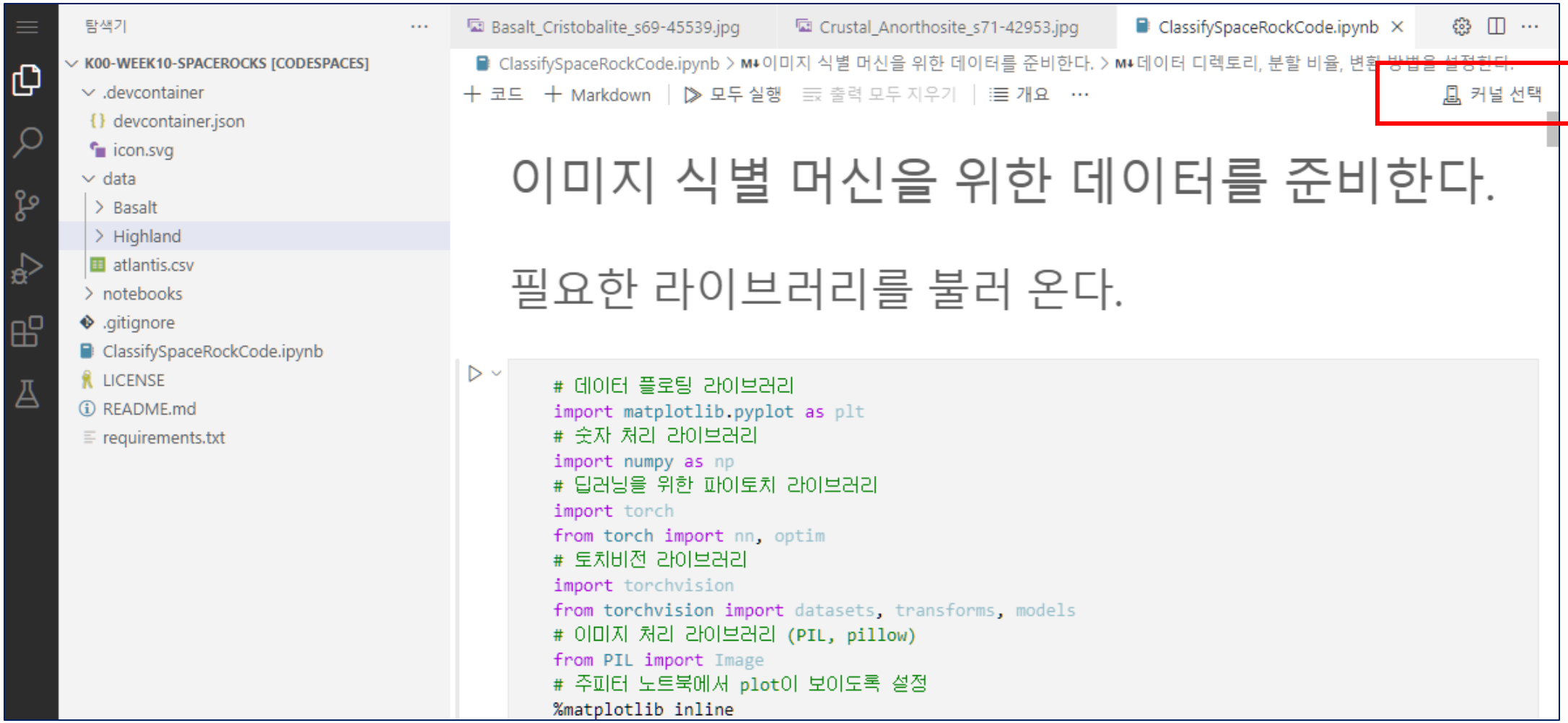
작업을 마친 후 Codespaces 페이지에서 가상머신을 중지(stop) 시킴으로써 core-hour를 아낄 수 있음





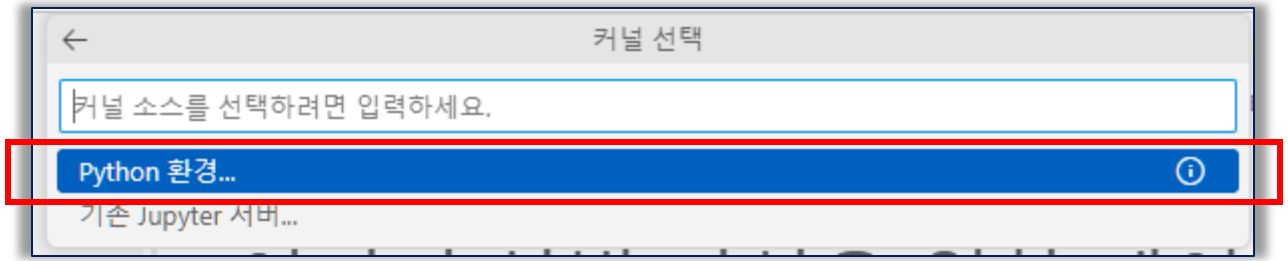
Jupyter Notebook 가상환경의 커널 선택

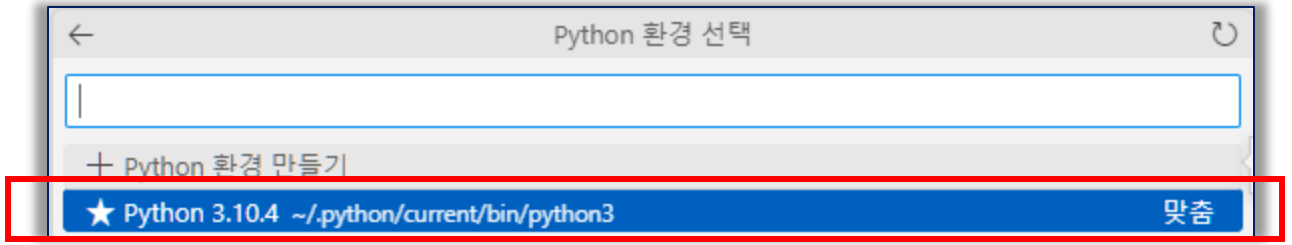
Jupyter Notebook의 사용 전에 개발환경을 위한 커널을 선택한다.



Jupyter Notebook 가상환경의 커널 선택

Jupyter Notebook의 사용 전에
개발 환경을 위한 커널을 선택한다.





커널이 Python 3.10.4 환경으로
지정된 것 확인 가능

목 차

파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

파이썬 코딩 실습 절차 확인

설정 및 데이터 로딩 함수 준비

필요한 라이브러리 import

데이터 디렉토리, 분할 비율, 변환 방법 설정

데이터 로딩 함수 작성

- 로딩 코드 연습
- 로딩 함수 작성

샘플 이미지 로딩 및 확인

모델링/평가 : FCL 수정

Compute device 결정(CPU/GPU)

ResNet50 모델 지정(사전학습)

FCL 설정(layer, 손실함수, 최적화함수)

모델링/평가 : FCL 학습/테스트

학습/테스트 변수 설정(에폭, 학습단계 등)

학습/검증/평가 절차 구현

- 평가 결과 확인

모델 저장

암석 이미지 예측

학습된 모델 불러오기

이미지 예측 함수 작성

이미지 예측

필요한 라이브러리 import

'출력 모두 지우기' 먼저 실행

이미지 식별 머신을 위한 데이터를 준비한다.

노트(markdown)

필요한 라이브러리를 불러 온다.

코드(python code)

실행

```
# 데이터 플로팅 라이브러리
import matplotlib.pyplot as plt
# 숫자 처리 라이브러리
import numpy as np
# 딥러닝을 위한 파이토치 라이브러리
import torch
from torch import nn, optim
# 토치비전 라이브러리
import torchvision
from torchvision import datasets, transforms, models
# 이미지 처리 라이브러리 (PIL, pillow)
from PIL import Image
# 주피터 노트북에서 plot이 보이도록 설정
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

[1]

데이터/이미지 시각화 라이브러리

숫자, 행렬 등의 계산 라이브러리

딥러닝을 위한 라이브러리

이미지 AI 모델을 위한 라이브러리

이미지 처리를 위한 라이브러리

주피터 노트북 환경을 위한 matplotlib.plot 설정

기본 변수 설정

데이터 위치, 분할 비율, 데이터 변환 방법을 설정

실행

이미지 데이터 디렉토리 ('.' 은 현재 디렉토리를 의미함)

학습 데이터와 검증(테스트) 데이터의 분할 비율(8:2)

이미지 데이터의 변환 (transform) 방법을 설정함

ResNet이 처리하는 이미지 크기(224x224)로 crop한 후 신경망에서 처리할 수 있도록 tensor로 변경함

기본 변수 설정

설정한 변수값을 확인함

(확인) 변환 방법을 출력하여 확인해 본다.

실행

```
# 설정한 이미지 데이터 변환 방법을 출력하여 확인한다.
print(t_transforms)
```

[3] ✓ 0.1s Python

```
Compose(
    RandomResizedCrop(size=(224, 224), scale=(0.08, 1.0), ratio=(0.75, 1.3333), interpolation=bilinear)
    Resize(size=224, interpolation=bilinear, max_size=None, antialias=None)
    ToTensor()
)
```

파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

데이터 로딩 함수 – 연습 : 데이터세트 폴더 등 지정

데이터 로더(loader) 생성을 위한 함수 작성을 위해 코드를 여러 블록으로 나누어 실행해 본다.

데이터를 로딩 함수를 작성한다.

(연습) trainloader와 testloader를 만들어 본다.

1. 학습 데이터 세트 및 테스트 데이터 세트의 디렉토리 및 변환 방식을 지정한다.

실행

```
# datasets.ImageFolder를 사용해서 학습 데이터(train_data)와 테스트 데이터(test_data)를 만든다.
train_data = datasets.ImageFolder(data_dir, transform=t_transforms)
test_data = datasets.ImageFolder(data_dir, transform=t_transforms)

# 학습 데이터의 형식을 확인한다.
print(train_data)

# 학습 데이터와 테스트 데이터의 길이를 확인한다.
print(len(train_data), len(test_data))
```

Python

데이터세트의 위치 및 변환 방법을 지정함

결과를 확인함

데이터 로딩 함수 – 연습 : 데이터세트 폴더 등 지정

데이터세트의 위치, 변환 방법, 및 데이터 개수 확인

```
# datasets.ImageFolder를 사용해서 학습 데이터(train_data)와 테스트 데이터(test_data)를 만든다.
train_data = datasets.ImageFolder(data_dir, transform=t_transforms)
test_data = datasets.ImageFolder(data_dir, transform=t_transforms)

# 학습 데이터의 형식을 확인한다.
print(train_data)

# 학습 데이터와 테스트 데이터의 길이를 확인한다.
print(len(train_data), len(test_data))
```

[4] ✓ 0.1s Python

```
Dataset ImageFolder
    Number of datapoints: 155
    Root location: ./data
    StandardTransform
Transform: Compose(
               RandomResizedCrop(size=(224, 224), scale=(0.08, 1.0), ratio=(0.75, 1.3333), interpolation=bilinear)
               Resize(size=224, interpolation=bilinear, max_size=None, antialias=None)
               ToTensor()
           )
155 155
```

data_dir 내에서 폴더명을 각 데이터들의 레이블로 처리할 수 있음

데이터 로딩 함수 – 연습 : 인덱스 섞기

2. 데이터세트를 섞기 위해, 우선 인덱스를 만들어 랜덤하게 섞는다.

```
# train_data 사이즈만큼의 정수값을 갖는 인덱스 리스트(indices)를 만들고 확인한다.
num_train = len(train_data)
indices = list(range(num_train))
print(indices)

# 인덱스 리스트를 랜덤하게 섞고 확인한다.
np.random.shuffle(indices)
print(indices)
```

[5] ✓ 0.1s

데이터 개수만큼 연속된 정수로 구성된(range) 인덱스 리스트(indices list)를 만든 후 출력

인덱스 리스트(indices list)를 임의로 섞은 후 출력

예시 :

index	data
0	A
1	B
2	C
3	D
4	E

index	data
3	D
0	A
1	B
4	E
2	C

Data를 랜덤하게 분할하는 효과

데이터 로딩 함수 – 연습 : 인덱스 섞기

2. 데이터세트를 섞기 위해, 우선 인덱스를 만들어 랜덤하게 섞는다.

실행

```
# train_data 사이즈만큼의 정수값을 갖는 인덱스 리스트(indices)를 만들고 확인한다.
num_train = len(train_data)
indices = list(range(num_train))
print(indices)

# 인덱스 리스트를 랜덤하게 섞고 확인한다.
np.random.shuffle(indices)
print(indices)
```

[5] ✓ 0.1s Python

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
[14, 76, 82, 129, 151, 104, 84, 74, 91, 43, 122, 147, 98, 4, 113, 73, 68, 146, 45, 17, 126, 118, 117, 154, 106, 40, 93,
```

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

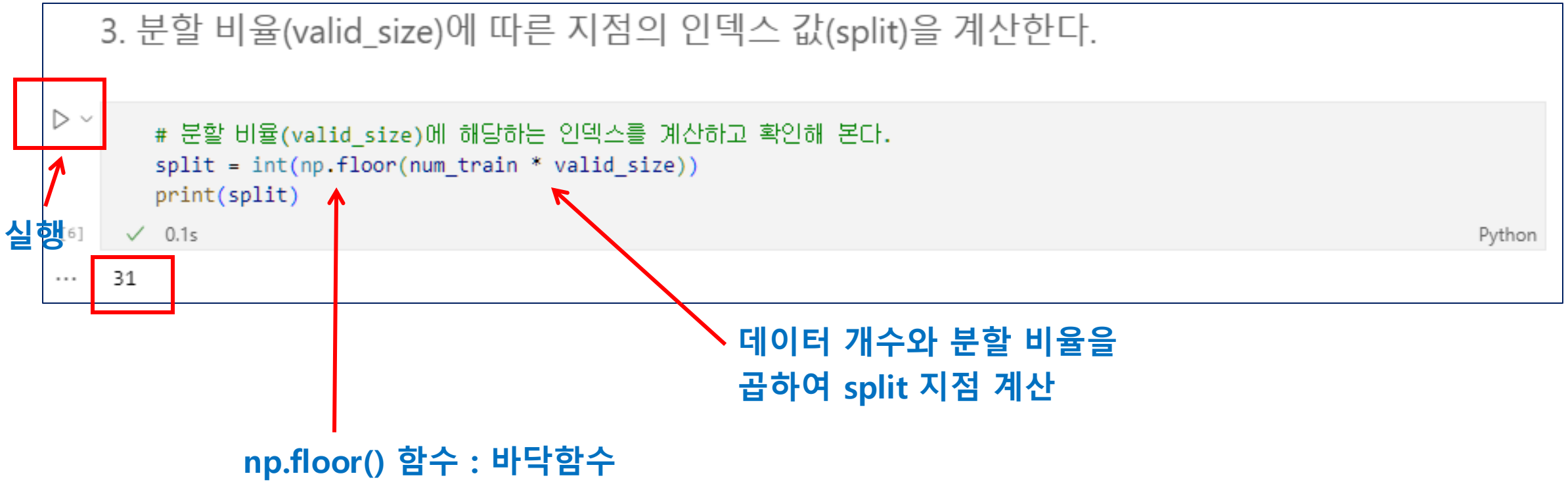
데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

데이터 로딩 함수 – 연습 : 분할 지점 계산

분할 비율(valid_size = 0.2)에 따른 분할 지점 계산



데이터 로딩 함수 – 연습 : 분할 지점 계산

4. split을 기준으로 학습 데이터 인덱스 리스트와 테스트 인덱스 리스트로 나눈다.

실행

```
# 학습 데이터 인덱스 리스트 및 테스트 인덱스 리스트를 만들고 확인해 본다.

train_idx, test_idx = indices[split:], indices[:split]

print(train_idx)
print(test_idx)
```

[7] ✓ 0.1s Python

```
[71, 46, 138, 23, 25, 121, 31, 132, 128, 10, 133, 107, 15, 124, 99, 142, 144, 141, 54, 5, 55, 105, 9, 69, 87, 65, 36, 5
[14, 76, 82, 129, 151, 104, 84, 74, 91, 43, 122, 147, 98, 4, 113, 73, 68, 146, 45, 17, 126, 118, 117, 154, 106, 40, 93,
```

Train_index와 test_index로 분할

파이썬 리스트 나누기 :

리스트명[시작 인덱스 : 끝 인덱스]

- 끝 인덱스는 포함하지 않음
- 시작 인덱스 또는 끝 인덱스 생략 가능

예) indices = [: 2]
→ [D, A]
예) indices = [2 :]
→ [B, E, C]

index	data
3	D
0	A
1	B
4	E
2	C

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

데이터 로딩 함수 – 연습 : 샘플러 및 로더

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

데이터 로딩 함수 – 데이터 로딩 함수 작성

코드들을 묶어서 load_split_train_test() 함수를 만든다.

실행

```
# 위의 코드들을 묶어서 load_split_train_test() 함수를 만든다. (입력 : 데이터 디렉토리, 분할 비율) (출력 : 학습 데이터 로더

def load_split_train_test(data_dir, valid_size) :
    t_transforms = transforms.Compose([
                transforms.RandomResizedCrop(224),
                transforms.Resize(224),
                transforms.ToTensor()
    ])

    train_data = datasets.ImageFolder(data_dir, transform=t_transforms)
    test_data = datasets.ImageFolder(data_dir, transform=t_transforms)
    num_train = len(train_data)
    indices = list(range(num_train))

    np.random.shuffle(indices)
    split = int(np.floor(num_train * valid_size))
    train_idx, test_idx = indices[split:], indices[:split]
    from torch.utils.data.sampler import SubsetRandomSampler

    train_sampler = SubsetRandomSampler(train_idx)
    test_sampler = SubsetRandomSampler(test_idx)

    trainloader = torch.utils.data.DataLoader(train_data, sampler=train_sampler, batch_size=16)
    testloader = torch.utils.data.DataLoader(test_data, sampler=test_sampler, batch_size=16)

    return trainloader, testloader
```

입력 : 데이터세트 폴더, 분할 비율

출력 : 학습 로더, 테스트 로더

[9] ✓ 0.1s Python

연습한 코드를 묶어서, 데이터 분할 및 샘플러를 포함하는 데이터 로딩 함수 작성

파이썬 함수 형식 :

```
def 함수명 (입력1, 입력2, ...) :
    statement ...
    return 출력1, 출력2, ...
```

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

데이터 로딩 함수 – 데이터 로딩 함수 확인

load_split_train_test() 함수를 이용하여 trainloader, testloader를 생성한다.

```
# load_split_train_test() 함수를 이용하여 trainloader와 testloader를 만들고 확인한다.
trainloader, testloader = load_split_train_test(data_dir, 0.2)

print(trainloader.dataset.classes)
print(testloader.dataset.classes)
```

[10] ✓ 0.1s Python

```
['Basalt', 'Highland']
['Basalt', 'Highland']
```

실행

작성한 함수를 사용하여
학습 로더와 테스트 로더 생성

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수

- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수

- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수

- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

샘플 이미지 로딩 및 확인 – 이미지/레이블 가져오기 함수

이미지 데이터 샘플들을 살펴본다.

임의의 데이터를 로딩한 후 이미지와 레이블을 반환하는 get_random_images() 함수를 만든다.

실행

```
def get_random_images(num) :

    data = datasets.ImageFolder(data_dir, transform=t_transforms)
    indices = list(range(len(data)))
    np.random.shuffle(indices)
    idx = indices[:num]

    from torch.utils.data.sampler import SubsetRandomSampler
    sampler = SubsetRandomSampler(idx)
    loader = torch.utils.data.DataLoader(data, sampler=sampler, batch_size=num)
    # loader에서 데이터를 한 개씩 꺼내 주는 iterator를 생성한다.
    dataiter = iter(loader)
    images, labels = dataiter.next()

    return images, labels
```

[11] Python

num : 가져올 이미지 개수

Indices[: num] : 지정한 개수 만큼 별도의 인덱스 리스트 생성

배치 size를 num으로 지정하여 데이터를 로드함

이미지와 레이블을 배치 size씩 순차적으로 가져옴

이미지들과 레이블들을 반환함

iterable 객체 : 데이터를 하나씩 가져올 수 있음

iter() : iterable 객체 생성

next() : 다음번 데이터를 하나 가져옴

샘플 이미지 로딩 및 확인 – 이미지/레이블 표시

임의 선택한 이미지를 표시해 본다.

실행

```python
# 5개의 이미지와 레이블을 랜덤하게 가져온다.
images, labels = get_random_images(5)
# 픽셀 배열을 PIL 형식의 이미지로 변환하고 이미지 크기를 지정한다.
to_pil = transforms.ToPILImage()
fig = plt.figure(figsize=(20, 20))
# 학습 데이터의 class 리스트를 얻는다.
classes = trainloader.dataset.classes
# 이미지를 표시하기 위한 설정을 한다.
for ii in range(len(images)) :
    image = to_pil(images[ii])
    sub = fig.add_subplot(1, len(images), ii+1)
    index = labels[ii].item()
    sub.set_title(classes[index])
    plt.axis('off')
    plt.imshow(image)
# 주피터 노트북에 이미지를 표시한다.
plt.show()
```

Python

5개의 이미지와 레이블을 가져옴

파이썬 이미지 형식(PIL)으로
변환하는 방식 및 크기 지정

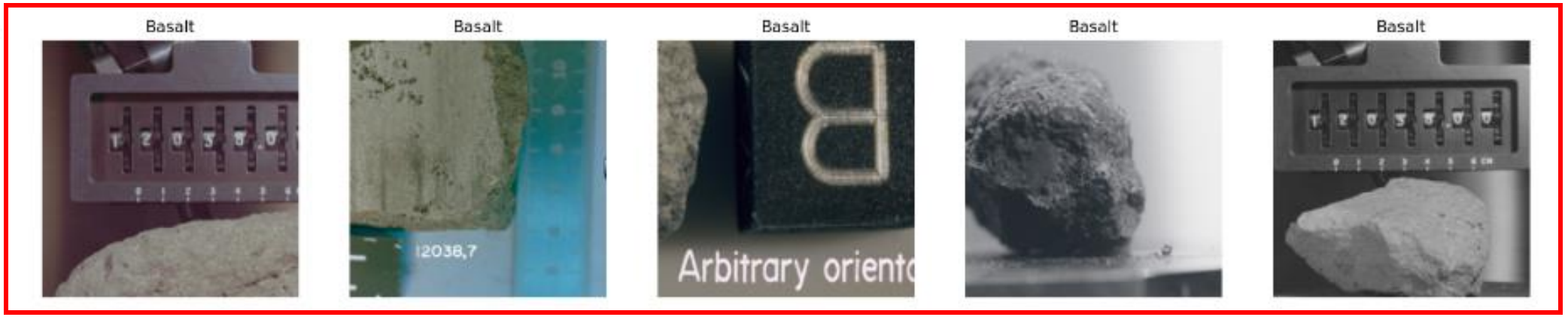
레이블이 추가된 이미지를 구성함

이미지를 표시함

샘플 이미지 로딩 및 확인 – 이미지/레이블 표시

```
# 학습 데이터의 class 리스트를 얻는다.
classes = trainloader.dataset.classes
# 이미지를 표시하기 위한 설정을 한다.
for ii in range(len(images)) :
    image = to_pil(images[ii])
    sub = fig.add_subplot(1, len(images), ii+1)
    index = labels[ii].item()
    sub.set_title(classes[index])
    plt.axis('off')
    plt.imshow(image)
# 주피터 노트북에 이미지를 표시한다.
plt.show()
```

[12] ✓ 1.7s Python



[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

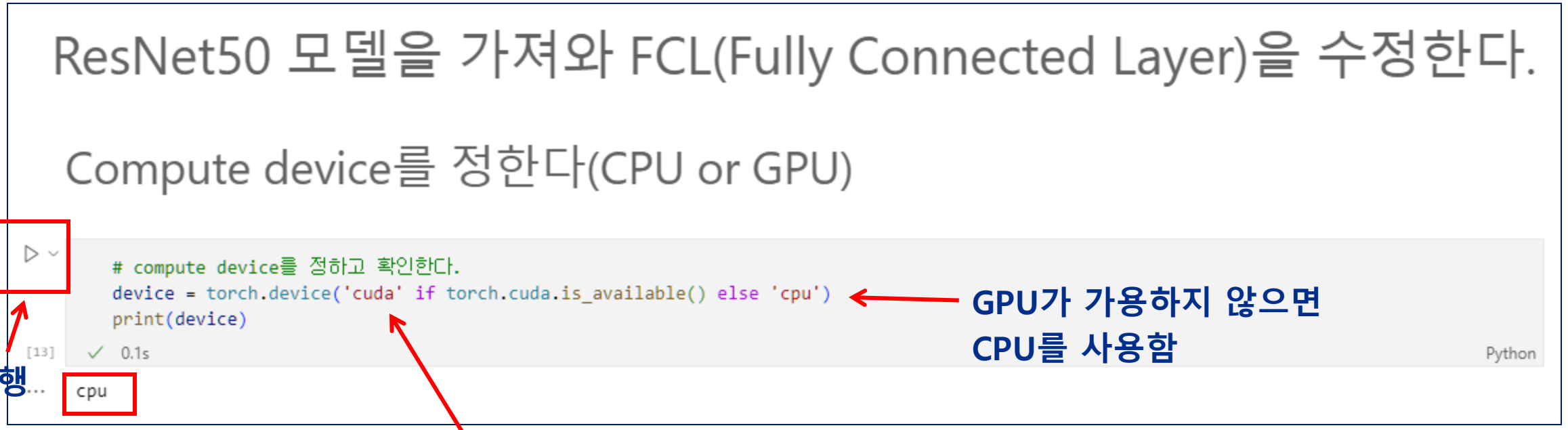
암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

Compute Device 확인



[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

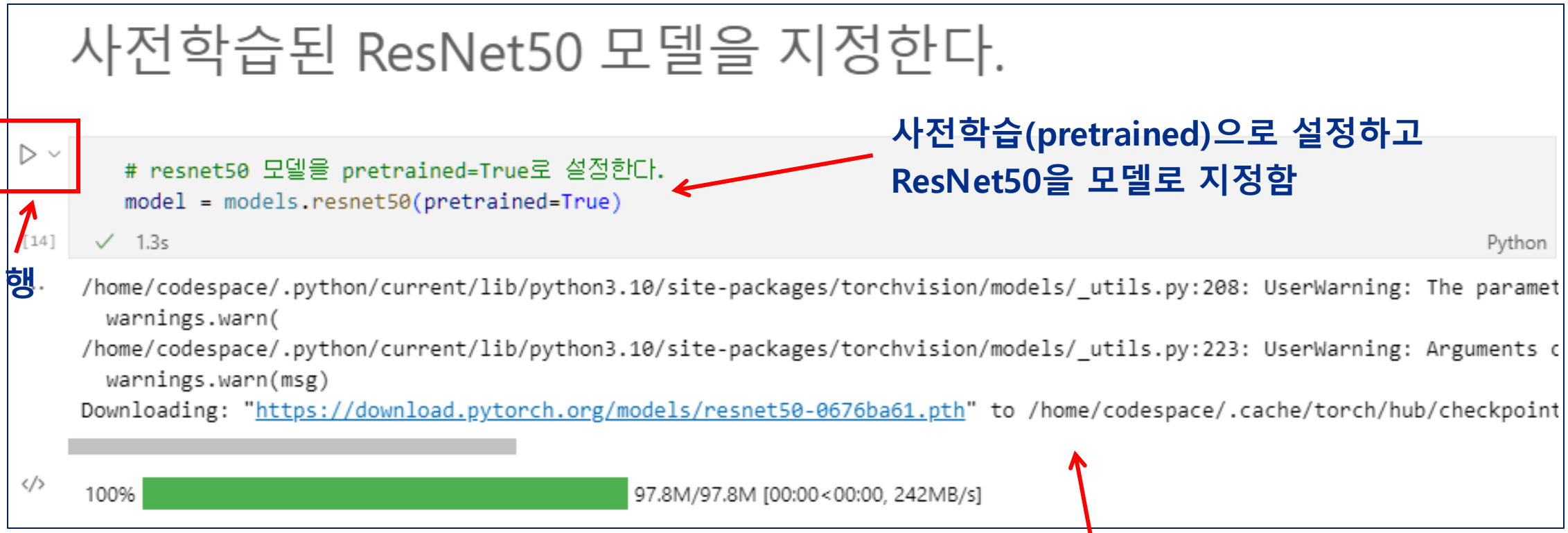
암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

ResNet50 모델 지정



[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

Fully Connected Layer(FCL) 수정 (전이학습)

FCL을 수정한다.(뉴런 구축, 신경망 연결, FCL의 layer 설정 등)

실행

```
# 모든 신경망 구축 : 전이학습을 위해 모델의 가중치를 freeze 한다.
for param in model.parameters():
    param.requires_grad = False
# 뉴런들을 연결하여 신경망을 생성한다.
model.fc = nn.Sequential(nn.Linear(2048, 512),
                         nn.ReLU(),
                         nn.Dropout(0.2),
                         nn.Linear(512, 2),
                         nn.LogSoftmax(dim=1))
# 손실함수를 Cross entropy loss 함수로 지정한다.
criterion = nn.NLLLoss()
# optimizer를 Adam으로 지정한다.
optimizer = optim.Adam(model.fc.parameters(), lr=0.003)
# 신경망을 compute device로 보낸다.
model.to(device)
# 종료 여부를 출력한다.
print('done!')
```

[16] ✓ 0.1s Python

```
done!
```

전이학습을 위해 파라미터 update 안함

FCL 수정 : (0) 2048 → 512 노드
(1) ReLU
(2) Dropout
(3) 512 → 2 노드
(4) LogSoftmax

손실함수 지정 : 교차 엔트로피 손실함수

최적화함수 : Adam 최적화함수

Compute device로 보냄

Fully Connected Layer(FCL) 수정 (전이학습)

수정한 FCL의 구조를 확인함

(확인) FCL을 확인해 본다.

```
print(model.fc)
```

[17] ✓ 0.1s

실행

```
Sequential(
  (0): Linear(in_features=2048, out_features=512, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.2, inplace=False)
  (3): Linear(in_features=512, out_features=2, bias=True)
  (4): LogSoftmax(dim=1)
)
```

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

FCL 학습 – 학습/테스트 변수 설정

에폭, 학습단계 등 학습/테스트를 위한 변수를 설정함

모델의 FCL을 학습시키고 테스트 한다.

모델 학습/검증을 위한 변수를 설정한다.

```
# 에폭 및 출력 간격을 설정한다.
epochs = 10
print_every = 5
# 손실 변수들을 초기화 한다.
running_loss = 0
train_losses, test_losses = [], []
# 현재의 학습 단계를 표현하는 steps 변수를 0으로 초기화 한다.
steps = 0
```

[32] Python

실행

10번의 에폭을 수행

5회의 배치마다 결과를 출력

변수들을 초기화 함

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

FCL 학습/테스트/평가

설정한 에폭만큼 모델을 학습시키며 검증/평가 한다.

실행 →

설정한 에폭 만큼 반복수행

학습 데이터에 대해 반복수행

순전파 수행 및 예측확률(로그) 계산

역전파 수행 (기울기 업데이트)

손실값 누적/계산

```
# 설정한 회수만큼 학습 후 테스트 및 평가해 본다.
for epoch in range(epochs) :
    # 에폭을 count 한다.
    epoch += 1
    # trainloader로부터 모든 이미지와 레이블을 로드한다.
    for inputs, labels in trainloader:
        # 학습 단계를 count 하고 출력한다.
        steps += 1
        print('Training step ', steps)
        # 입력 데이터(이미지, 레이블)를 device로 보낸다.
        inputs, labels = inputs.to(device), labels.to(device)
        # 기존에 학습된 gradient 값을 초기화 한다.(이전에 학습한 값이 영향을 주지 않도록 함)
        optimizer.zero_grad()
        # 입력 데이터로 순전파를 수행하고 로그 확률을 얻는다.
        logps = model.forward(inputs)
        # 손실(loss) 값들을 계산한다.
        loss = criterion(logps, labels)
        # 손실값을 이용하여 gradient를 update한다.
        loss.backward()
        # gradient를 이용하여 설정된 optimizer로 파라미터를 update한다.
        optimizer.step()
        # 손실값을 누적/계산한다.
        running_loss += loss.item()
```

데이터(이미지/레이블)를 device로 전달

기울기 값 초기화

손실값 계산

최적화함수로 파라미터 업데이트

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

FCL 학습/테스트

```
# 학습 단계 5회마다 모델을 테스트/평가 한다.
if steps % print_every == 0:
    # 손실과 정확도 변수를 초기화 한다.
    test_loss = 0
    accuracy = 0
    # 모델 평가 모드로 전환한다.
    model.eval()
    # 모델 평가 시 gradient를 계산하지 않도록 한다.
    with torch.no_grad():
        # testloader로부터 모든 이미지와 레이블을 로드한다.
        for inputs, labels in testloader:
            # 입력 데이터(이미지, 레이블)를 device로 보낸다.
            inputs, labels = inputs.to(device), labels.to(device)
            # 입력 데이터로 순전파를 수행하고 로그 확률을 얻는다.
            logps = model.forward(inputs)
            # 손실(loss) 값들을 계산한다.
            batch_loss = criterion(logps, labels)
            # 손실값을 누적시킨다.
            test_loss += batch_loss.item()
            # 로그 확률로부터 진짜 확률값을 계산한다.
            ps = torch.exp(logps)
            # 가장 큰 확률값과 class를 얻는다. (topk : k번째로 큰 값)
            top_p, top_class = ps.topk(1, dim=1)
            # 레이블들을 top_class와 동일한 형태로 바꾼 후 같은 값들을 얻는다.
            equals = top_class == labels.view(*top_class.shape)
            # equals를 float 텐서로 바꾼 후 평균 정확도를 누적/계산한다.
            accuracy += torch.mean(equals.type(torch.FloatTensor)).item()
    # 학습 손실값과 테스트 손실값을 추가한다.
    train_losses.append(running_loss/len(trainloader))
    test_losses.append(test_loss/len(testloader))
    # 학습 손실값, 테스트 손실값, 테스트 정확도를 출력한다.
    print("Epoch {}/{}: ".format(epoch, epochs),
          "Train loss: {:.3f}.. ".format(running_loss/print_every),
          "Test loss: {:.3f}.. ".format(test_loss/len(testloader)),
          "Test accuracy: {:.3f}\n".format(accuracy/len(testloader)))
    # running_loss 값을 초기화 한다.
    running_loss = 0
    # 모델 학습 모드로 전환한다.
    model.train()
    break
```

학습 횟수가
5의 배수일때마다
모델 테스트/평가 수행

모델 평가 모드로 전환
(테스트/검증 수행)

테스트 시 기울기를
업데이트하지 않도록 함

테스트 데이터세트에
대해 반복수행

순전파 수행 후
로그 확률 예측

손실값(오차) 계산

손실값(오차) 누적

로그 확률을
원래의 확률값으로 변경

확률값으로부터 class 예측

맞게 예측한 결과를
equals에 할당함

예측 결과를 float로
변경 후 평균 정확도를
누적 계산

학습 오차와
테스트 오차 저장

학습 오차,
테스트 오차,
정확도를 출력

모델 학습 모드로
전환

FCL 학습/테스트/평가 – 결과 확인

5 step마다 오차 및 정확도 출력

```
···   Output exceeds the size limit. Open the full output data in a text editor
      Training step  1
      Training step  2
      Training step  3
      Training step  4
      Training step  5
      Epoch 1/10:  Train loss: 2.611..  Test loss: 1.345..  Test accuracy: 0.615

      Training step  6
      Training step  7
      Training step  8
      Training step  9
      Training step  10
      Epoch 2/10:  Train loss: 1.712..  Test loss: 0.934..  Test accuracy: 0.446

      Training step  11
      Training step  12
      Training step  13
      Training step  14
      Training step  15
      Epoch 3/10:  Train loss: 0.984..  Test loss: 0.430..  Test accuracy: 0.777

      Training step  16
      Training step  17
      Training step  18
      Training step  19
      ...
      Training step  49
      Training step  50
      Epoch 10/10:  Train loss: 0.306..  Test loss: 0.271..  Test accuracy: 0.842
```

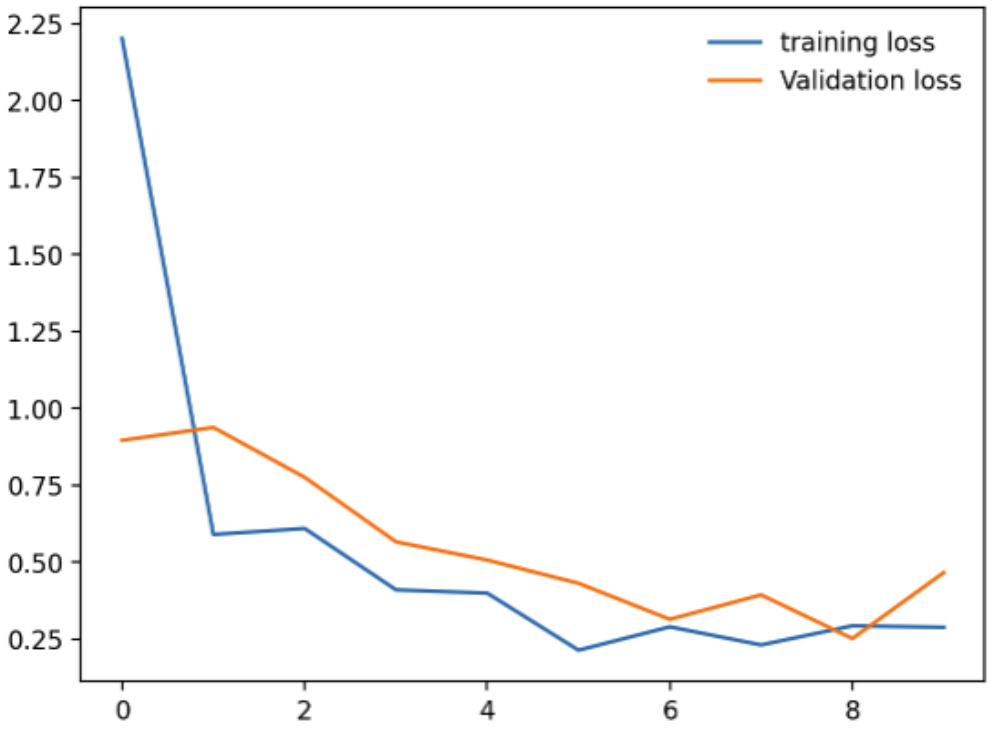
에폭이 증가함에 따라 오차는 줄고 정확도가 높아짐

FCL 학습/테스트/평가 – 결과 확인

(확인) 학습 손실값과 테스트 손실값을 그래프로 확인한다.

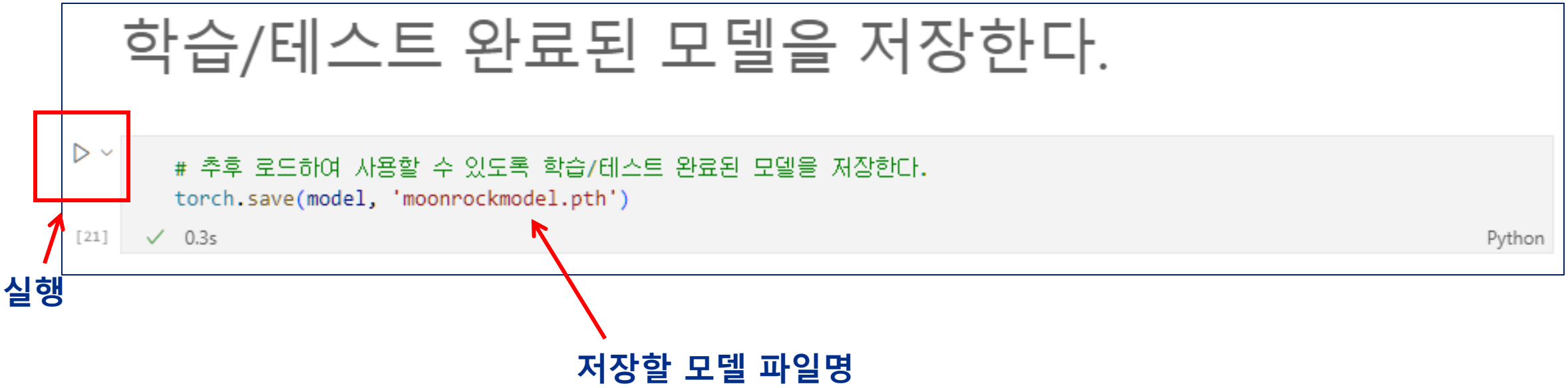
```
%matplotlib inline
%config InlineBackend.figure_format='retina'

plt.plot(train_losses, label='training loss')
plt.plot(test_losses, label='Validation loss')
plt.legend(frameon=False)

# in this graph, what is x-axis? y-axis?
# x-axis: epoch
# y-axis: loss
```

```
<matplotlib.legend.Legend at 0x7fb4b55e6620>
```



FCL 학습/테스트/평가 – 결과 확인

학습/테스트 완료된 모델을 저장하면 추후 불러 와서 사용 가능함



목 차

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수
- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수
- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수
- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

학습된 모델 불러오기

이미지 예측을 위해 학습/테스트 완료된 모델을 load 한다.

완성된 모델을 사용하여 예측한다.

저장한 모델을 불러온다.

```
[26]
# 저장한 모델을 불러온다.
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = torch.load('moonrockmodel.pth')
```

실행

(확인) 불러온 모델을 확인해 본다.

```
print(model)
```

학습된 모델 불러오기

```
...   Output exceeds the size limit. Open the full output data in a text editor
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
...
    (3): Linear(in_features=512, out_features=2, bias=True)
    (4): LogSoftmax(dim=1)
  )
)
```

불러온 모델 구조 확인

2 class 구분을 위한 layer

[참고] 파이썬 코드 구조

준비
- 라이브러리 import, 디렉토리, 분할 비율, 변환 방법 등 설정

학습/테스트 데이터 로더 생성
Compute device 설정
모델 설정 : ResNet50 (pretrained)
신경망 FCL 수정
- FCL 입력, 출력층 설정
- 활성화함수, 손실함수, 최적화함수

모델 학습/검증/테스트
- 변수 설정 및 초기화
- 에폭만큼 반복 수행
  - 학습 데이터들에 대해 반복 수행
    - 순전파 및 손실(loss) 계산
    - 역전파 및 비용(cost) 계산 (최적화 함수 적용)
    - 5회마다 테스트 및 평가
      - 테스트 데이터들에 대해 반복 수행
        - 순전파를 통해 비용 계산
        - Class 예측 및 정확도 계산
- 학습 및 테스트 손실값 확인(그래프)

암석 예측
- 5개의 이미지를 불러와서 예측

데이터 로더 생성 함수

- 이미지 폴더, 변환 방법 지정
- 데이터 세트 랜덤 분할(학습, 테스트)
- 학습/테스트 데이터 로더 반환

이미지 가져오기 함수

- 이미지 데이터 섞기
- 지정된 개수만큼 순차적으로 로드
- 이미지와 레이블 튜플 반환

이미지 예측 함수

- 이미지를 텐서로 변환
- 모델을 통해 class 예측 (확률이 큰 값)

이미지 예측 함수 작성

이미지 예측 함수 작성 : predict_image()

이미지 데이터 예측

5개의 이미지를 임의로 가져와 예측해 본다.

실행

```
# 모델 평가 모드로 전환한다.
model.eval()
# 5개의 이미지를 랜덤하게 가져온 후 PIL 형식 변환, 표시할 이미지 크기를 설정한다.
to_pil = transforms.ToPILImage()
images, labels = get_random_images(5)
fig = plt.figure(figsize=(20, 20))
# 데이터의 class 목록을 얻는다.
classes = trainloader.dataset.classes

# 5개의 이미지에 대해 loop를 수행한다.
for ii in range(len(images)):
    # 각 이미지에 대해 class를 예측한다.
    image = to_pil(images[ii])
    index = predict_image(image)
    # 이미지 아래에 class를 표시하도록 설정한다.
    sub = fig.add_subplot(1, len(images), ii+1)
    res = labels[ii].item() == index
    sub.set_title(classes[index] + ':' + str(res))
    plt.axis('off')
    plt.imshow(image)
# 레이블이 추가된 이미지를 보여준다.
plt.show()
```

[27] ✓ 1.9s Python

이미지 5개 가져오기

각 이미지에 대해 예측

각 이미지의 화면 표시 설정

결과 보여주기

이미지 데이터 예측

[27] ✓ 1.9s Python

...