

Instruction for Vision end-to-end Project

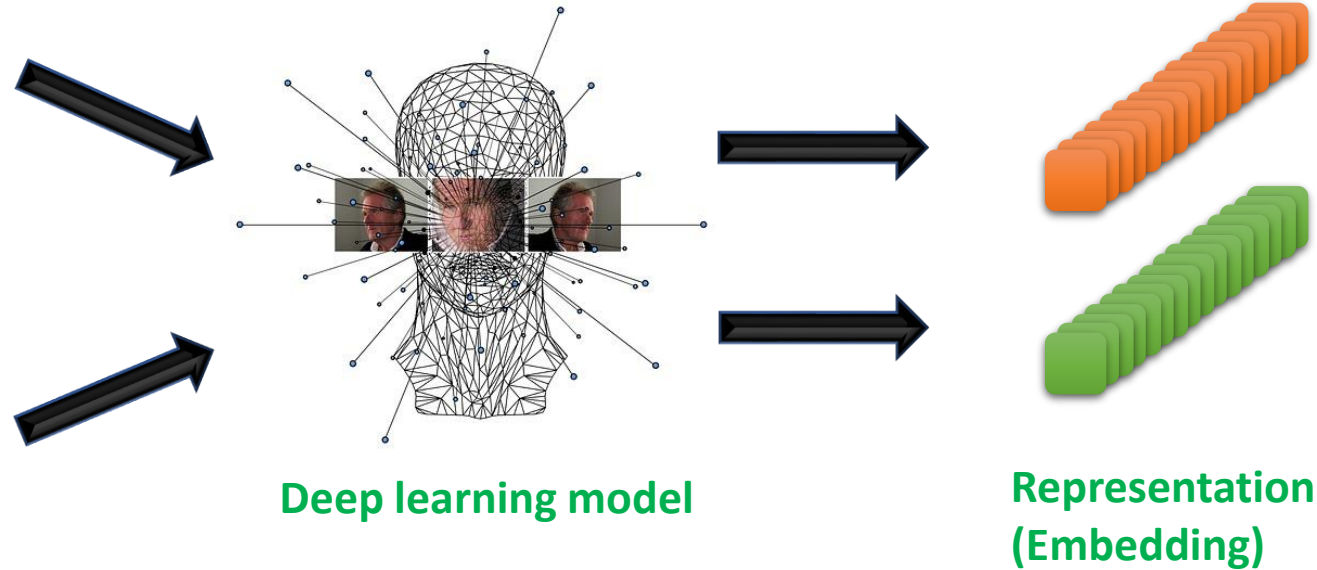
JIHO AN

Outline

- Introduction
- Background
- Overall process
- Project structure
- Setting
- Running script

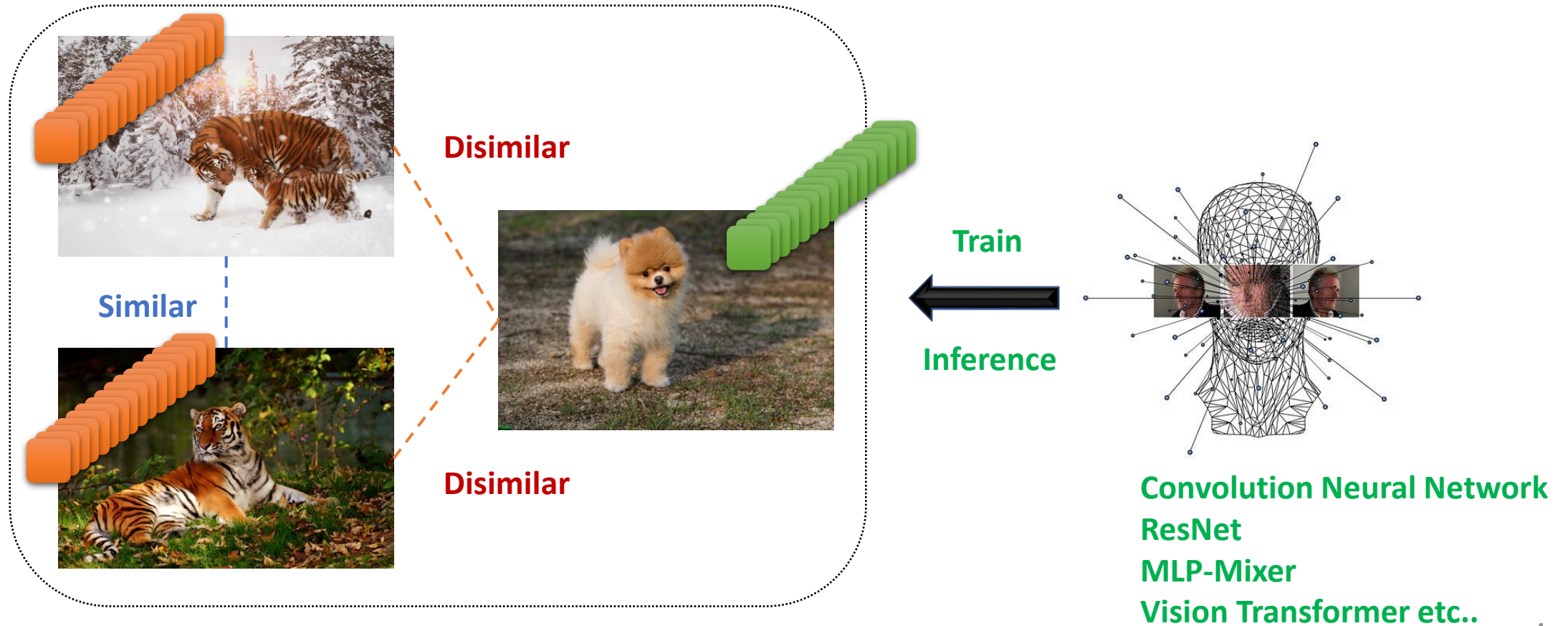
Introduction

- ✓ 본 프로젝트는 자기지도 학습(Self-supervised learning) 중 하나인 대조 학습(Contrastive learning)을 이용하여, 이미지 간의 유사성을 학습하는 것을 주제로 함.
 - 표상 학습(Representation learning)을 수행



Introduction

- ✓ 본 프로젝트는 자기지도 학습(Self-supervised learning) 중 하나인 대조 학습(Contrastive learning)을 이용하여, 이미지 간의 유사성을 학습하는 것을 주제로 함.



Background

- ✓ 머신러닝 및 딥러닝 모델을 구축하고 학습하는 과정에서, 그 행위의 목적은 개인적인 연구가 대부분을 차지하였음.
- ✓ 이에 따라, 주로 로컬 환경에서 Visual Studio code, Pycharm, Jupyter notebook을 사용하여 개발을 진행함.
- ✓ 물론, 가끔씩 라이브러리 의존성 관계에 따른 충돌이 발생하였으나, 해당 프로젝트 폴더에 접근하는 사람이 없으므로, 전반적으로 큰 애로 사항은 없었음.
- ✓ But, 연구실 내에서 점점 다른 사람들과 코드를 공유할 일이 많아지게 되었고, 서로 다른 장비를 사용하기에 같은 코드를 돌리더라도 누구는 Error가 나고, 또다른 사람은 멀쩡히 코드가 동작하여 난처한 상황이 자주 발생하였음.

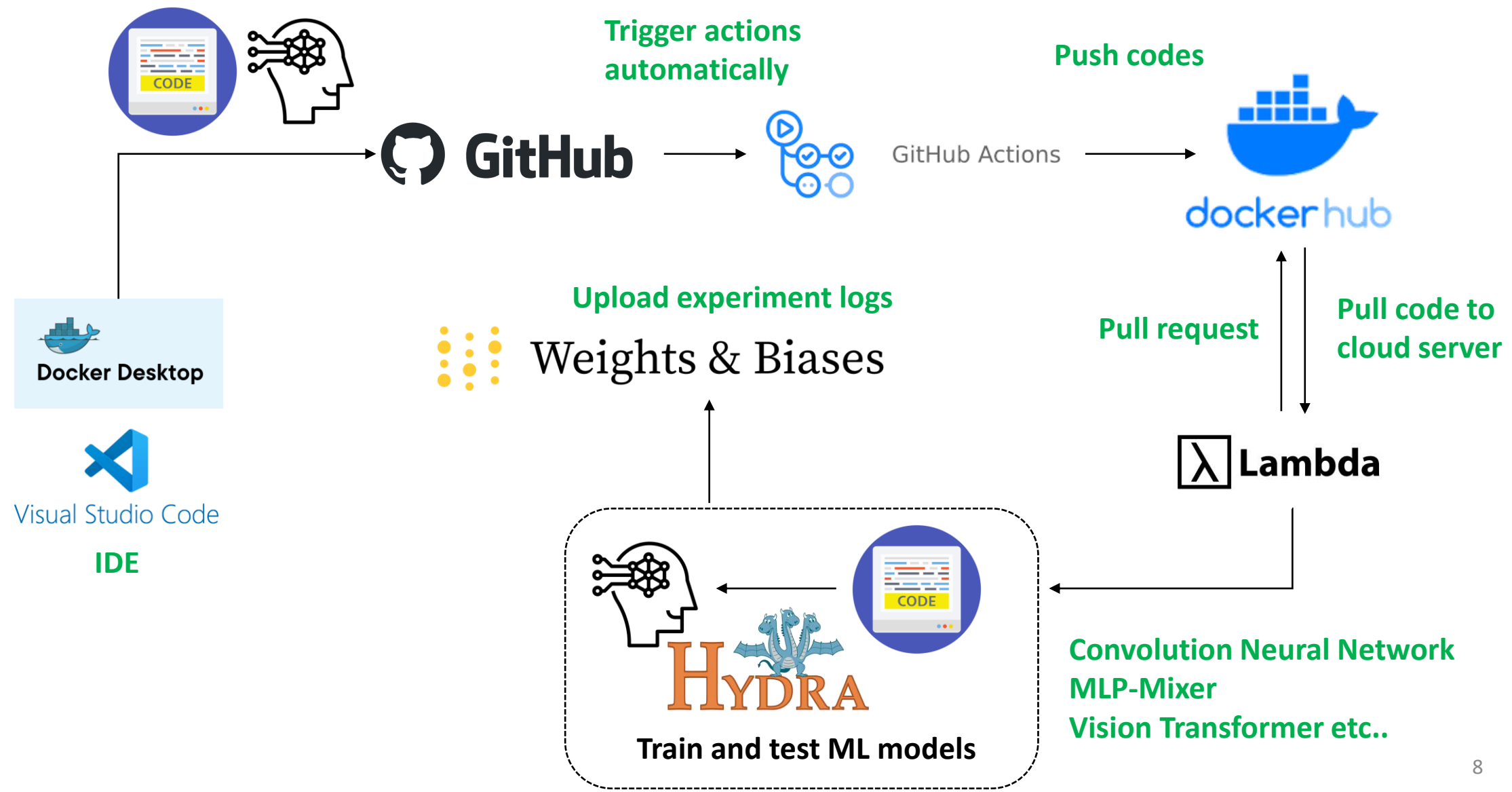
Background

- ✓ 또한, 국책과제 / 경진대회 등 다양한 협업 활동을 하는 과정에서, 최대한 코드를 간결하게 정리하여 공유해야 일의 효율성을 높일 수 있다는 생각을 '뼈저리게' 하였고, 아래의 팁을 배우게 됨.
 - Tip 1: 기능(모듈) 단위로 파일을 작성
 - Tip 2: 최대한 파일 간 중복성을 피하는 방향으로 코드를 작성
 - Tip 3: 개발 완료시 주기적으로 테스트용 코드(또는 모델)을 공유하고 피드백을 통해 지속적으로 코드를 발전시킬 것

Background

- ✓ 따라서, 본 프로젝트는 이미지 간 유사성 학습을 주제로 아래의 사항들을 구현하기로 함.
 - 코드와 설정의 분리: 설정을 별도의 파일로서 보관
 - Hydra 및 Hydra-zen을 통해 동적으로 설정 파일을 생성하고, 이에 따라 모델을 관리함.
 - 프로젝트 자동 배포
 - Github에 프로젝트를 등록할 때마다, 자동으로 해당 프로젝트의 모든 내용을 Docker hub에 배포
 - Dockerfile을 통한 프로젝트 관리
 - 코드 및 프로젝트 실행에 필요한 패키지 정보를 저장함으로써, 사용자의 환경과 무관하게 동작이 가능
- 모델 생성 및 학습 & 평가부터 시작하여, 프로젝트 배포까지 한 번에 이루어지는 End-to-End 시스템을 개발하는 것이 궁극적인 목표!!!!

Project Overall Process



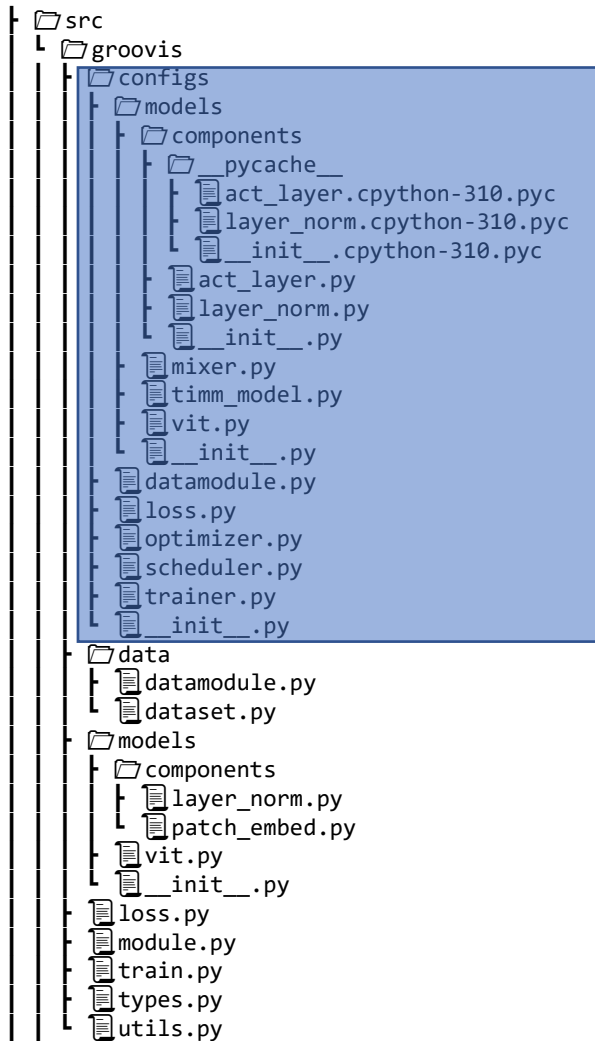
Project Overall Process

- ① IDLE: Visual studio code에서 제공하는 extension 중 Remote controller를 사용하여, Docker desktop과 연동함으로써, Linux 환경에서 코드를 개발하고 github repository에 개발 내용을 push.
- ② Github action: Git bash 커맨드를 통해 프로젝트 폴더를 Github에 push할 경우, [Github actions](#)에서 프로젝트 폴더를 [컨테이너화\(Containerization\)](#)하여, Docker hub에 자동으로 push.
- ③ Docker hub: Github action을 통해 push된 image가 존재하는 곳.
- ④ Cloud server(Optional): 본 프로젝트에서는 Lambda labs에서 instance를 대여하여 모델을 학습하고, 그 과정 및 결과를 Weight & Biases에 실시간으로 업로드 함.

Project structure

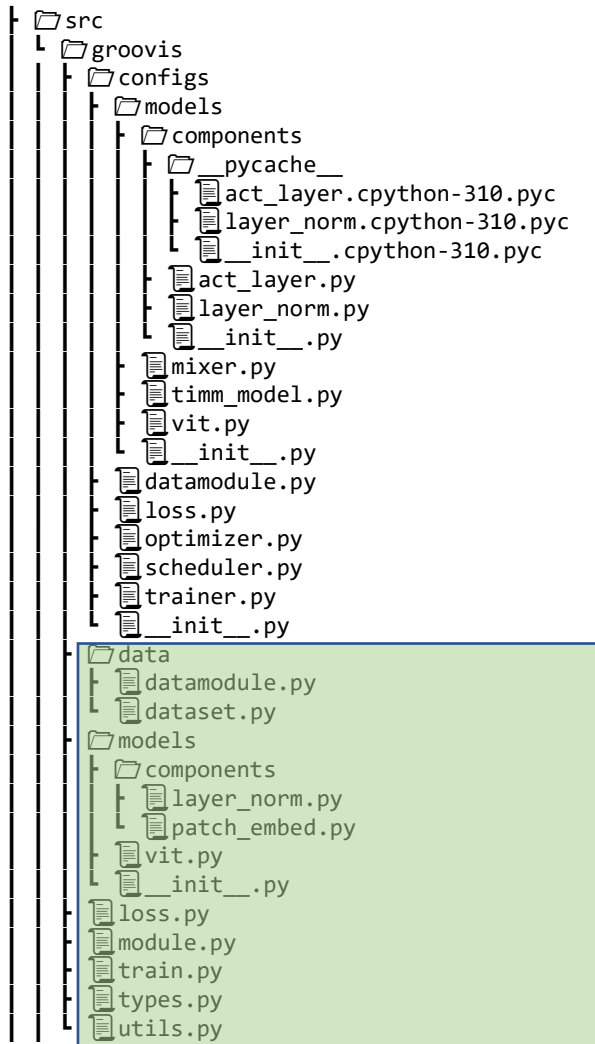
- ✓ 본 프로젝트는 크게 아래의 폴더들로 구성됨.
 - .devcontainer: 로컬 환경에서 Visual studio code를 통해 Docker desktop과 연동할 때, Docker container를 생성하는 역할 (Optional)
 - .github/workflow: 로컬 환경에서 수정된 프로젝트 내용을 Github에 push할 때, Github action이 어떠한 역할을 수행할 지 알려주는 역할 (Optional)
 - src: 본 프로젝트의 핵심 코드(실험 설정, 모델, 모델 학습 및 평가, 로깅 등)를 포함
 - wandb: 실험 내용을 실시간으로 기록하고, 이를 Weights & Biases 홈페이지에 동기화하기 위하여 필요한 내용을 저장

Project structure(src folder)



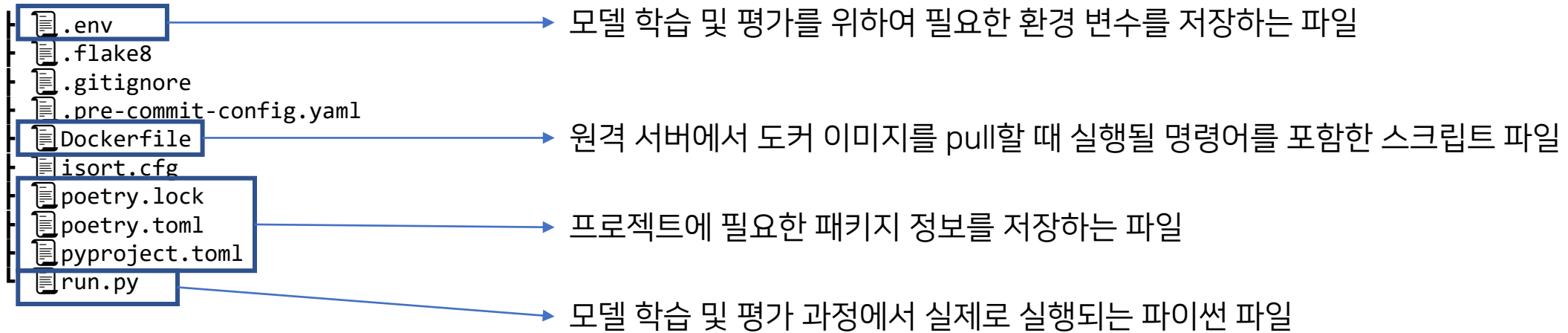
- Convolutional Neural Network(CNN) / MLP-Mixer / Parallel Vision Transformer -> Model
- Dataset / DataLoader / Datamodule -> Data
- Trainer / Optimizer / Scheduler -> Trainer
- 해당 모듈에 대한 설정을 별도로 보관함.
 - ✓ Hydra 및 Hydra-zen을 통해 런타임 과정에서 자동으로 yaml 파일 생성
 - ✓ 생성된 yaml 파일의 설정에 부합하는 클래스 인스턴스를 생성하여 코드를 실행

Project structure(src folder)



- Data / Trainer / Model / Embedding 등 실험에 필요한 실제 구현체를 포함.

Project structure(Other files)

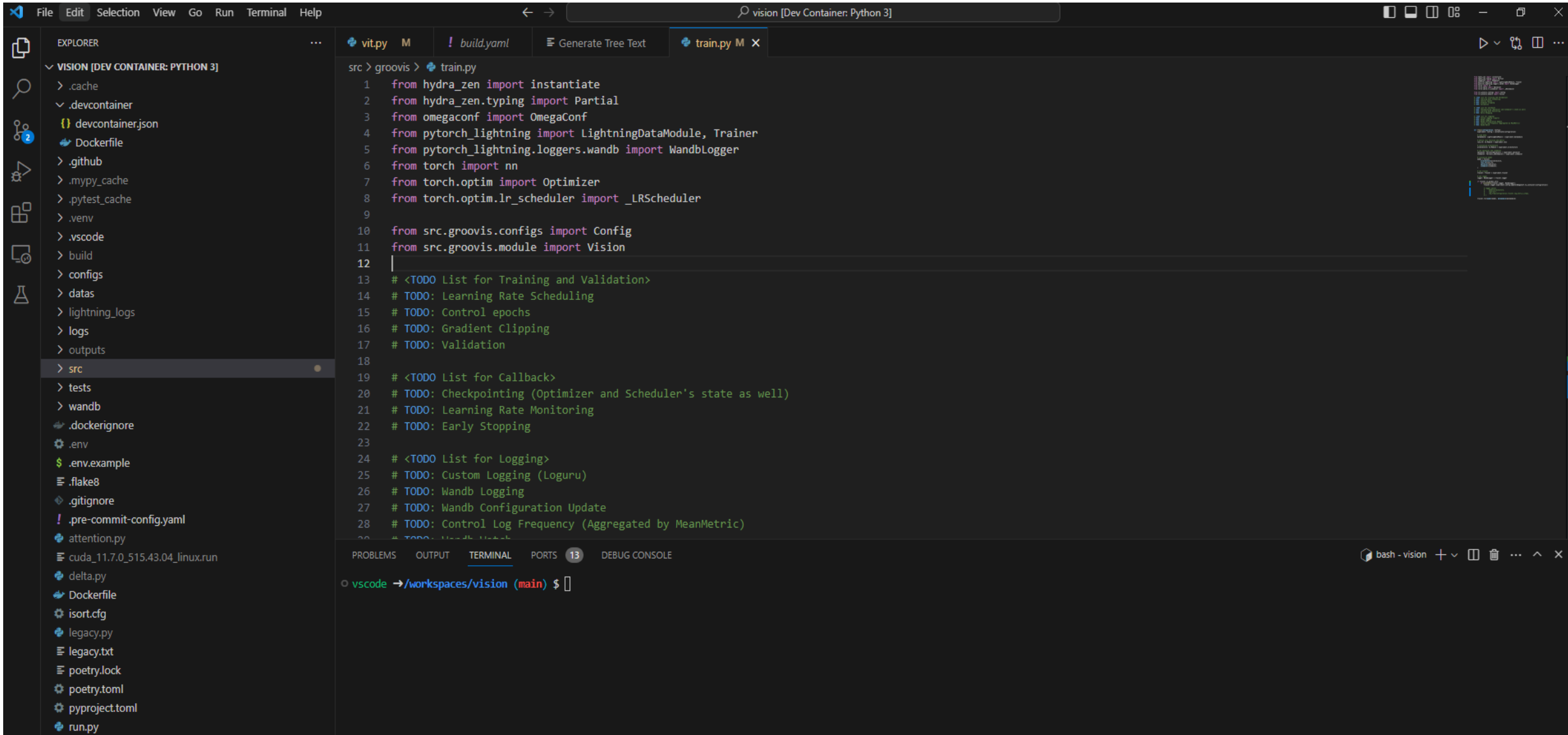


Setting(IDLE) -> Optional

- ✓ 로컬 환경, 특히 윈도우 운영체제에서는 **WSL2**을 통해 Linux 커맨드를 손쉽게 사용할 수 있음.
- ✓ 또한, **Docker desktop**을 통해 윈도우에서 간단하게 도커 환경에 접속이 가능함.
- ✓ Visual studio 개발 툴에서 **Dev Containers**을 설치하면, docker desktop과의 연동이 가능함.

- ① Install WSL2 in powershell + Install Docker desktop
- ② Intall Visual studio code
- ③ Install Dev Containers
- ④ Open Project folder in Container using Dev Containers

Setting(IDLE) -> Optional



Setting(Using GPU in Docker desktop) -> Optional

- ✓ Docker desktop에서는 호스트의 GPU를 컨테이너 위에서 사용할 수 있도록하는 기능을 제공하고 있음.
- ✓ 아래의 설정을 순서대로 완료할 것!
 - WSL2(Docker desktop)에서 GPU 활용 가이드: <https://docs.nvidia.com/cuda/wsl-user-guide/index.html>
- ① 로컬 호스트에서 사용하는 GPU 버전에 맞는 Nvidia driver 설치
- ② 컨테이너 상에서 Cuda toolkit 설치 -> gcc, cuda, torch 호환성 확인할 것
 - 참고 1: <https://developer.nvidia.com/cuda-toolkit-archive>
 - 참고 2: <https://jjuke-brain.tistory.com/entry/GPU-서버-사용법-CUDA-PyTorch-버전-맞추기-총정리>

Setting(Using GPU in Docker desktop) -> Optional

③ CUDA 라이브러리 파일 등록

- `$ export PATH=$PATH:/usr/local/{YOUR_CUDA_VERSION}/bin`
- `$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/{YOUR_CUDA_VERSION}/lib64`
- 참고: <https://change-words.tistory.com/entry/linux-LDLIBRARYPATH>

④ Cuda container toolkit 설치

- 참고: <https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html>

⑤ Dev containers 상에서 GPU 사용을 가능하게 하기 위한 추가 설정

- 참고: <https://stackoverflow.com/questions/72129213/using-gpu-in-vs-code-container>
- .devcontainer/devcontainer.json 파일에 반영 완료함!!

Setting(Using GPU in Docker desktop) -> Optional

② 컨테이너 상에서 Cuda toolkit 설치 -> gcc, cuda, torch 호환성 확인할 것

- 본 프로젝트에서 OS는 Debian GNU/Linux 11, CUDA versio은 11.7을 사용함.

Operating System	Linux	Windows							
Architecture	x86_64	ppc64le	arm64-sbsa						
Distribution	CentOS	Debian	Fedora	OpenSUSE	RHEL	Rocky	SLES	Ubuntu	WSL-Ubuntu
Version	11								
Installer Type	deb (local)	deb (network)	runfile (local)						

Download Installer for Linux Debian 11 x86_64

The base installer is available for download below.

>Base Installer

Installation Instructions:

```
$ wget https://developer.download.nvidia.com/compute/cuda/11.7.0/local_installers/cuda_11.7.0_515.43.04_linux.run
$ sudo sh cuda_11.7.0_515.43.04_linux.run
```

본 명령어를 프롬프트 상에서
순차적으로 실행할 것!!!

Setting(Using GPU in Docker desktop) -> Optional

- ✓ GPU 설정을 완료하여, nvidia-smi 및 nvcc -v(또는 nvcc -version) 명령을 통해 호스트의 Nvidia driver와 런타임 CUDA 라이브러리가 정상적으로 인식되는지 확인할 것.

```
vscode →/workspaces/vision (main) $ nvidia-smi
Thu Sep 14 08:59:32 2023
+-----+
| NVIDIA-SMI 530.30.02                Driver Version: 531.18          CUDA Version: 12.1          |
+-----+-----+-----+-----+-----+-----+
| GPU   Name                               Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf              Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                               |                      |              MIG M.   |
+-----+-----+-----+-----+-----+-----+
|  0  NVIDIA GeForce RTX 3060              On | 00000000:01:00.0  On |          N/A         |
|  0%   43C   P8              18W / 170W | 1018MiB / 12288MiB |    7%      Default   |
|                               |                      |              N/A     |
+-----+-----+-----+-----+-----+-----+

Processes:
+-----+-----+-----+-----+-----+-----+
| GPU   GI   CI          PID   Type   Process name                      GPU Memory |
|   ID   ID   ID                      |           Usage |
+-----+-----+-----+-----+-----+-----+
| No running processes found |
+-----+-----+-----+-----+-----+-----+

```

Host에서 인식된 Nvidia driver의 정보

```
vscode →/workspaces/vision (main) $ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Tue_May__3_18:49:52_PDT_2022
Cuda compilation tools, release 11.7, V11.7.64
Build cuda_11.7.r11.7/compiler.31294372_0
vscode →/workspaces/vision (main) $

```

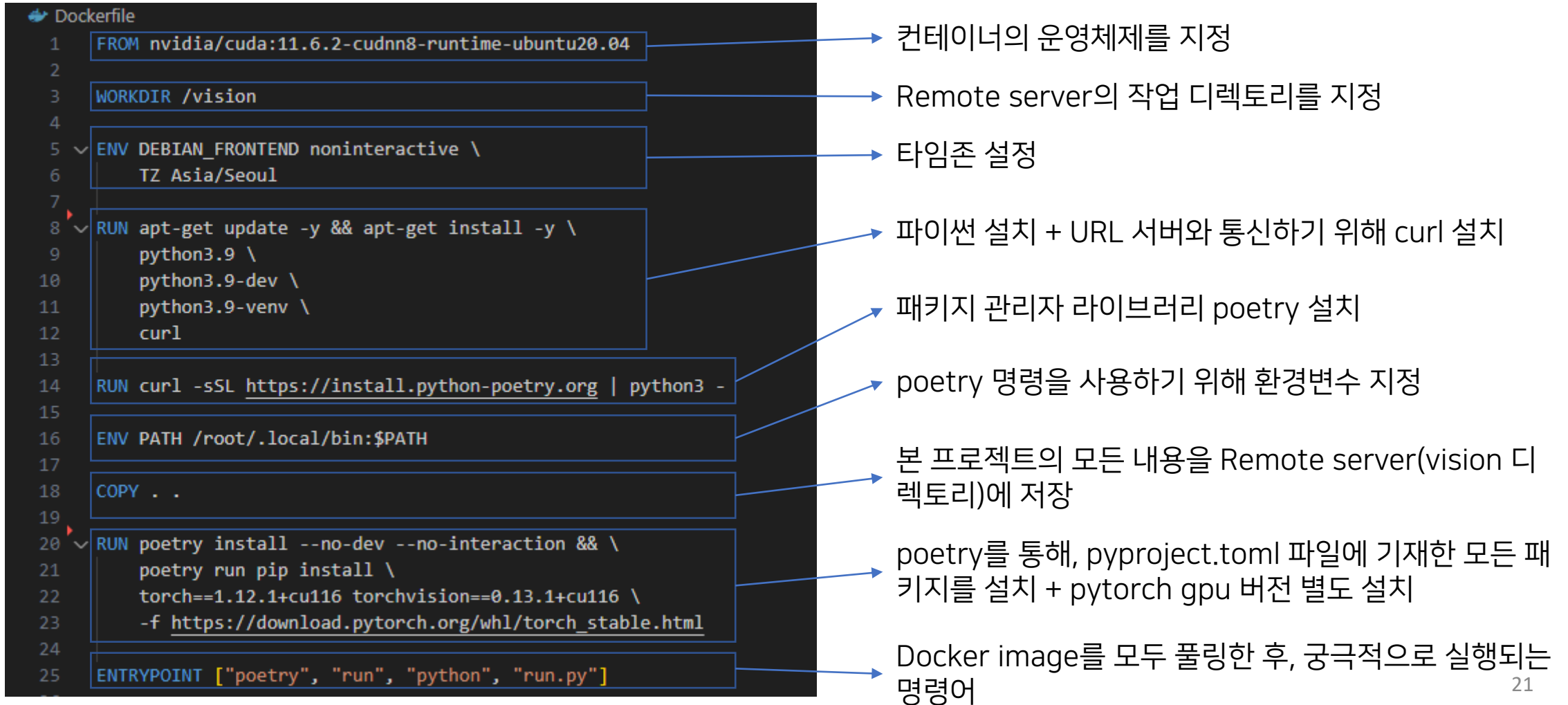
런타임시 GPU 구동을 위해 사용할 CUDA 버전

Setting(Dev containers)

- ✓ Visual studio code에서 Dev containers extension을 통해 Docker desktop에 연동할 경우, '.devcontainer' 폴더에 있는 ['Dockerfile'](#)과 ['devcontainer.json'](#)에 적시한 내용을 기반으로 도커 컨테이너를 생성함.
- ✓ 윈도우 운영체제에서 Docker desktop을 연동하지 않는 이상, 프로젝트에서 작성한 '.devcontainer' 폴더 내에 Dockerfile과 devcontainer.json에 있는 내용은 docker container 생성 시 반영되지 않음.
 - 세부 사항은 각 파일의 주석을 참고할 것.
- ✓ 다른 환경에서 프로젝트를 실행할 경우, [vision 디렉토리에 있는 Dockerfile](#)을 통해 본 프로젝트에 필요한 라이브러리 및 코드가 자동으로 설치되게끔 설정함!

Setting(Dockerfile)

- ✓ 윈도우 이외의 다른 환경에서 본 프로젝트를 수행할 경우 실행되는 Dockerfile의 내용은 다음과 같음.



```
1 FROM nvidia/cuda:11.6.2-cudnn8-runtime-ubuntu20.04
2
3 WORKDIR /vision
4
5 ENV DEBIAN_FRONTEND noninteractive \
6     TZ Asia/Seoul
7
8 RUN apt-get update -y && apt-get install -y \
9     python3.9 \
10    python3.9-dev \
11    python3.9-venv \
12    curl
13
14 RUN curl -sSL https://install.python-poetry.org | python3 -
15
16 ENV PATH /root/.local/bin:$PATH
17
18 COPY . .
19
20 RUN poetry install --no-dev --no-interaction && \
21     poetry run pip install \
22     torch==1.12.1+cu116 torchvision==0.13.1+cu116 \
23     -f https://download.pytorch.org/whl/torch_stable.html
24
25 ENTRYPOINT ["poetry", "run", "python", "run.py"]
```

컨테이너의 운영체제를 지정

Remote server의 작업 디렉토리를 지정

타임존 설정

파이썬 설치 + URL 서버와 통신하기 위해 curl 설치

패키지 관리자 라이브러리 poetry 설치

poetry 명령을 사용하기 위해 환경변수 지정

본 프로젝트의 모든 내용을 Remote server(vision 디렉토리)에 저장

poetry를 통해, pyproject.toml 파일에 기재한 모든 패키지를 설치 + pytorch gpu 버전 별도 설치

Docker image를 모두 풀링한 후, 궁극적으로 실행되는 명령어

Setting(Library)

- ✓ 로컬 환경에서 직접 본 프로젝트 구동에 필요한 라이브러리를 다운받고자 할 경우, 아래의 명령어를 실행.
 - \$ poetry install
- ✓ 라이브러리 목록은 'pyproject.toml' 파일에 기재가 되어 있음.

```
[tool.poetry.dependencies]
python = ">=3.9,<3.11"
torch = "1.12.1"
einops = "^0.6.1"
albumentations = "^1.2.1"
torchvision = "0.13.1"
python-dotenv = "^1.0.0"
omegaconf = "^2.3.0"
loguru = "^0.7.0"
wandb = "^0.15.8"
datasets = "^2.14.4"
pytorch-lightning = "^1.7.7"
torchmetrics = "^0.9.3"
torch-optimizer = "^0.3.0"
rich = "12.6.0"
timm = "0.6.11"
hydra-core = "^1.2.0"
hydra-zen = "v0.9.0rc5"
jaxtyping = "^0.2.21"
beartype = "^0.15.0"
flake8 = "^4.0.1"
```

```
[tool.poetry.group.dev.dependencies]
flake8 = "^4.0.1"
black = "^23.7.0"
pytest = "^7.4.0"
imagehash = "^4.3.1"
pre-commit = "^3.3.3"
mypy = "0.982"

[tool.black]
line-length = 88

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

Setting(Github)

- ✓ 로컬에서 프로젝트 폴더와 본인의 깃허브 repository를 연동하여 업데이트 된 내용을 원격 전송할 경우, 아래의 명령어를 사용 .
 - \$ git init -> Git 초기화
 - \$ git branch -M main -> branch 이름을 main으로 변경
 - \$ git remote add origin "your github ssh address" -> 현재 작업 폴더와 해당 github repository를 연동
 - \$ git add . -> 현재 폴더에서 업데이트 된 내용을 git에 추가
 - \$ git commit -m "message" -> 업데이트 된 내용을 git repository에 commit
 - \$ git push origin main -> commi된 내용을 실제 github repository에 반영(push)
 - git add, git commit, git push 명령어는 반드시 순차적으로 실행해야 함.

Setting(Github Actions)

- ✓ 로컬에서 프로젝트 폴더와 본인의 깃허브 repository를 연동하여 업데이트 된 내용을 원격 전송할 경우, 아래 파일에 의해 github action이 자동으로 trigger됨.
 - .github/workflows/build.yaml
 - 자세한 내용은 해당 파일의 주석을 참고할 것
- ✓ 본 프로젝트에서 사전 작성한 build.yaml은 크게 세 가지의 job으로 구성이 되어 있음.
 - ✓ Test: 파이썬, poetry, 본 프로젝트에 필요한 패키지 설치 (캐싱 포함)
 - ✓ free-disk-space: 불필요한 파일을 삭제 (Github actions에서 할당된 메모리 용량을 초과하는 오류를 피하기 위함)
 - ✓ build: 해당 프로젝트 폴더를 도커 이미지로 build 하고, 이를 docker hub에 자동으로 등록

Setting(Github Actions)

✓ .github/workflows/build.yaml 파일 중 'build' job의 경우 몇 가지 신경써야 할 사항이 있음

- ① name: Login to Docker Hub
- ② name: Build and push

```
84 build:
85   runs-on: ubuntu-latest
86   needs: free-disk-space
87   steps:
88     - name: Checkout
89       uses: actions/checkout@v3
90
91     - name: Set up QEMU
92       uses: docker/setup-qemu-action@v2
93
94     - name: Set up Docker Buildx
95       uses: docker/setup-buildx-action@v2
96
97     - name: Login to Docker Hub
98       uses: docker/login-action@v2
99       with:
100         username: ${ secrets.DOCKERHUB_USERNAME }}
101         password: ${ secrets.DOCKERHUB_TOKEN }}
102
103     - name: Build and push
104       uses: docker/build-push-action@v3
105       with:
106         context: .
107         push: true
108         tags: jihoahn9303/vision:latest
109         cache-from: type=gha
110         cache-to: type=gha,mode=max
111
```

Docker Hub에 등록한 본인 계정에 접속하기 위한 설정

Docker image를 빌드하고 Docker Hub에 등록하기 위한 설정

Setting(Github Actions)

① name: Login to Docker Hub

- ✓ build.yaml을 통해 자동으로 Docker hub에 등록된 계정에 접근하기 위해서는 아래 두 가지의 환경 변수 설정이 필요함.

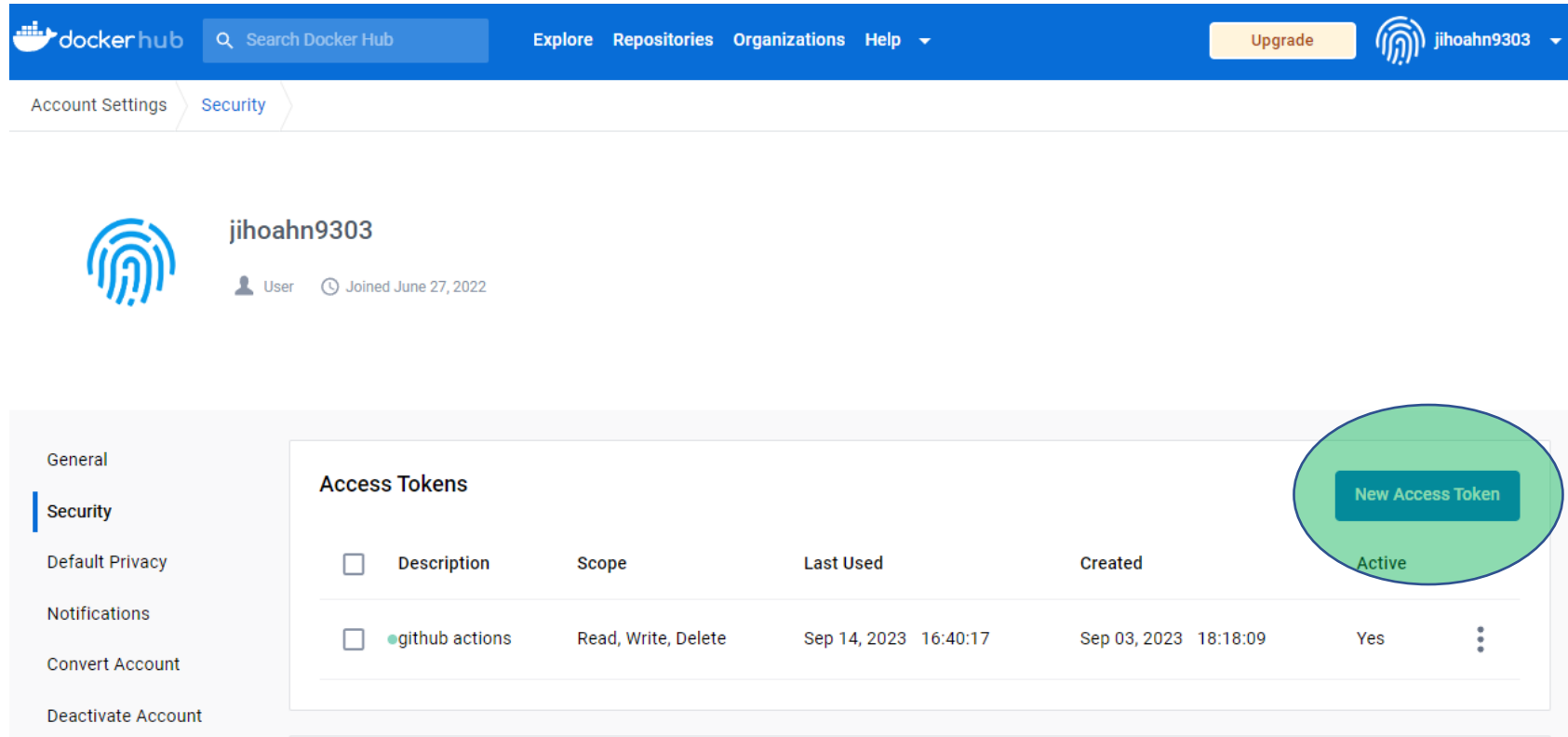
- `${{ secrets.DOCKERHUB_USERNAME }}`
- `${{ secrets.DOCKERHUB_TOKEN }}`

- ✓ 이를 위하여 해야 할 설정은 다음과 같음.
 - Docker hub에서 token을 발급
 - 발급 받은 docker hub token과 id를 github repository에 등록

Setting(Github Actions)

① name: Login to Docker Hub: Docker hub에서 token을 발급

1. Docker hub > Account settings > Security를 클릭
2. 'New Access Token' 클릭



The screenshot shows the Docker Hub interface for user 'jihoahn9303'. The 'Security' tab is selected in the left sidebar. The 'Access Tokens' section is visible, showing a table with one token named 'github actions'. A green circle highlights the 'New Access Token' button in the top right corner of the 'Access Tokens' section.

Access Tokens

<input type="checkbox"/>	Description	Scope	Last Used	Created	Active	
<input type="checkbox"/>	github actions	Read, Write, Delete	Sep 14, 2023 16:40:17	Sep 03, 2023 18:18:09	Yes	⋮

Setting(Github Actions)

① name: Login to Docker Hub: Docker hub에서 token을 발급

3. 토큰 발급 -> 발급 받은 토큰이 `${{ secrets.DOCKERHUB_TOKEN }}`의 값임!!!!

- 발급 받는 그 화면에서만 토큰 값을 확인할 수 있으므로, 해당 토큰 값을 안전한 곳에 저장해놓을 것!!!!

New Access Token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access Token Description *
Hello_docker

Access permissions
Read, Write, Delete

Read, Write, Delete tokens allow you to manage your repositories.

Cancel Generate

원하는 이름으로 설정 가능

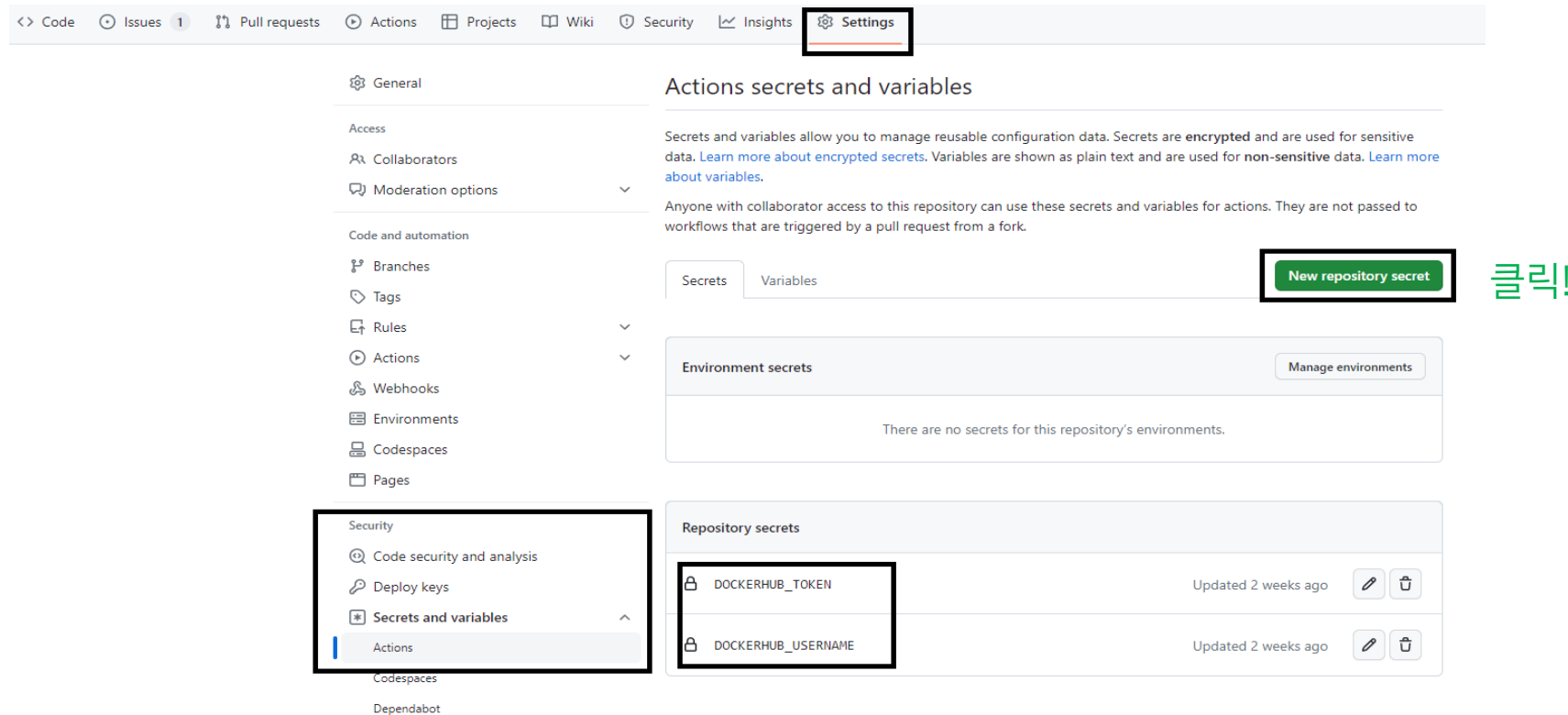
권한은 Read, Write, Delete 모두 설정

설정 완료 후 클릭

Setting(Github Actions)

① name: Login to Docker Hub: Github repository에 발급 받은 토큰 등록

- 본인의 Github repository 내에서 아래 그림과 같이 이동
- Settings > Security > Secrets and variables > Actions



Setting(Github Actions)

① name: Login to Docker Hub: Github repository에 발급 받은 토큰 등록

- New repository secret에 두 가지 값을 입력

1. DOCKERHUB_TOKEN

Actions secrets / New secret

Name *

DOCKERHUB_TOKEN

반드시 이름 변경하지 말 것!!!

Secret *

docker hub에서 발급 받은 토큰 값

Add secret

Setting(Github Actions)

① name: Login to Docker Hub: Github repository에 발급 받은 토큰 등록

- New repository secret에 두 가지 값을 입력

2. DOCKERHUB_USERNAME

Actions secrets / New secret

Name *

DOCKERHUB_USERNAME 반드시 이름 변경하지 말 것!!!

Secret *

DOCKER HUB 계정 이름

Add secret

Setting(Github Actions)

- ① name: Login to Docker Hub: Github repository에 발급 받은 토큰 등록
- 정상적으로 등록이 되면 아래 그림처럼 Repository secrets에 두 가지의 토큰 목록이 출력.

The screenshot displays the GitHub repository settings page. The top navigation bar includes tabs for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Settings' tab is selected. The left sidebar shows the 'General' section expanded, with 'Access', 'Collaborators', and 'Moderation options' visible. The 'Code and automation' section is also expanded, showing 'Branches', 'Tags', 'Rules', 'Actions', 'Webhooks', 'Environments', 'Codespaces', and 'Pages'. The 'Security' section is expanded, showing 'Code security and analysis', 'Deploy keys', 'Secrets and variables', and 'Actions'. The 'Secrets and variables' section is highlighted. The main content area shows the 'Actions secrets and variables' section. The 'Secrets' tab is selected, and a green button labeled 'New repository secret' is visible. The 'Repository secrets' section lists two secrets: 'DOCKERHUB_TOKEN' and 'DOCKERHUB_USERNAME', both updated 2 weeks ago. The text '클릭!' (Click!) is written in green next to the 'New repository secret' button.

Setting(Github Actions)

② name: Build and push

- tags에 있는 값을 반드시 변경할 것
- tags 형식: {본인 docker hub 계정명}/{docker image 이름}:{tag 이름}

```
84 build:
85   runs-on: ubuntu-latest
86   needs: free-disk-space
87   steps:
88     - name: Checkout
89       uses: actions/checkout@v3
90
91     - name: Set up QEMU
92       uses: docker/setup-qemu-action@v2
93
94     - name: Set up Docker Buildx
95       uses: docker/setup-buildx-action@v2
96
97     - name: Login to Docker Hub
98       uses: docker/login-action@v2
99       with:
100         username: ${ secrets.DOCKERHUB_USERNAME }}
101         password: ${ secrets.DOCKERHUB_TOKEN }}
102
103     - name: Build and push
104       uses: docker/build-push-action@v3
105       with:
106         context: .
107         push: true
108         tags: jihoahn9303/vision:latest
109         cache-from: type=gna
110         cache-to: type=gha,mode=max
111
```

Running script(Docker)

- ✓ Remote server에서 Docker hub에 등록된 이미지를 가져와서 사용하면 됨.
 - 본인이 직접 Docker image를 배포한 경우, 본인의 hub 주소를 사용할 것
 - hub 주소 형식: {Dockerhub 계정}/{Image 이름}:{Tag 이름} (Tag 이름의 경우 필수 사항은 아님)

```
ubuntu@104-171-202-41:~$ sudo docker pull jihoahn9303/vision:latest
latest: Pulling from jihoahn9303/vision
56e0351b9876: Already exists
0e353182dfa4: Already exists
63add13c711b: Already exists
1210b79751b0: Already exists
eb1e2ff09225: Already exists
f33aea389e5c: Already exists
048de3511c8c: Already exists
f08c685ff0ca: Already exists
3052b6588483: Already exists
25b49978466a: Already exists
bfb1646fcc27: Already exists
286212146d83: Already exists
00a7019e2164: Already exists
5ffc6ee4da00: Pull complete
02ed59ecff4a: Pull complete
Digest: sha256:9607d19c8bb93d05eb8e3344cfffbf83e801004b44c1841cd7ecdb579b7824817
Status: Downloaded newer image for jihoahn9303/vision:latest
docker.io/jihoahn9303/vision:latest
```

Running script(Docker)

- ✓ Remote server에서 다음과 같이 스크립트 파일을 생성하여 모델 학습 및 평가가 가능함.
 - 설정 값은 알맞게 변경하여 사용할 것 -> 본 프로젝트에서는 'run.sh'로 파일명을 지정
 - 'python run.py --help' 명령어를 통해 변경 가능한 설정 값 조회 가능!
 - .env 파일을 미리 서버에 저장해놓을 것 (템플릿은 Github 홈페이지 Usage 파트를 참고)

```
docker run \
  --env-file=.env \
  --volume=/home/ubuntu/ViT-imagenet \
  --gpus=all \
  --ipc=host \
  --tty \
  jihoahn9303/vision \
  datamodule/dataset=imagenet \
  architecture=parallel_vit_base \
  trainer.max_epochs=3 \
  trainer.strategy="ddp_find_unused_parameters_false" \
  trainer.logger.group=Mixer-exp \
  trainer.logger.name=Parallel-ViT-1 \
  optimizer.lr=0.0005
```

Running script(Docker)

- ✓ 아래의 명령어를 통해 스크립트 파일을 실행할 경우, 데이터 다운로드부터 모델 학습 및 평가 과정을 수행함.
 - `$ sudo sh run.sh`
- ✓ Hugging face로부터 ImageNet 데이터 다운로드

```
distributed_backend=nccl
All distributed processes registered. Starting with 8 processes
-----

[2023-09-15 04:01:36,679][torch.distributed.distributed_c10d][INFO] - Rank 1: Completed store-based barrier for key:store_based_barrier_key:1 with 8 nodes.
[2023-09-15 04:01:36,681][torch.distributed.distributed_c10d][INFO] - Rank 7: Completed store-based barrier for key:store_based_barrier_key:1 with 8 nodes.
Downloading builder script: 100%|██████████████████████████████████████████████████████████████████████████████| 5.05k/5.05k [00:00<00:00, 11.1MB/s]
Downloading metadata: 100%|██████████████████████████████████████████████████████████████████████████████| 28.6k/28.6k [00:00<00:00, 13.8MB/s]
Downloading readme: 100%|██████████████████████████████████████████████████████████████████████████████| 85.4k/85.4k [00:00<00:00, 13.6MB/s]
Downloading extra modules: 100%|██████████████████████████████████████████████████████████████████████████████| 46.4k/46.4k [00:00<00:00, 26.5MB/s]
Downloading data files: 0%|██████████████████████████████████████████████████████████████████████████████| 0/3 [00:00<?, ?it/s^
Downloading data: 100%|██████████████████████████████████████████████████████████████████████████████| 29.1G/29.1G [11:03<00:00, 43.9MB/s]
Downloading data: 100%|██████████████████████████████████████████████████████████████████████████████| 29.3G/29.3G [08:03<00:00, 60.5MB/s]
Downloading data: 100%|██████████████████████████████████████████████████████████████████████████████| 29.0G/29.0G [09:35<00:00, 50.5MB/s]
Downloading data: 100%|██████████████████████████████████████████████████████████████████████████████| 29.2G/29.2G [08:54<00:00, 54.7MB/s]
Downloading data: 100%|██████████████████████████████████████████████████████████████████████████████| 29.1G/29.1G [09:39<00:00, 50.3MB/s]
Downloading data: 100%|██████████████████████████████████████████████████████████████████████████████| 6.67G/6.67G [01:42<00:00, 65.0MB/s]
Downloading data: 100%|██████████████████████████████████████████████████████████████████████████████| 13.5G/13.5G [04:16<00:00, 52.7MB/s]
Downloading data files: 100%|██████████████████████████████████████████████████████████████████████████████| 3/3 [53:20<00:00, 1066.71s/it]
Generating train split: 100%|██████████████████████████████████████████████████████████████████████████████| 1281167/1281167 [16:54<00:00, 1263.39 examples/s]
Generating validation split: 100%|██████████████████████████████████████████████████████████████████████████████| 50000/50000 [00:41<00:00, 1203.46 examples/s]
Generating test split: 53%|██████████████████████████████████████████████████████████████████████████████| 52772/100000 [00:41<00:31, 1495.75 examples/s]
```

Running script(Docker)

- ✓ 학습할 모델의 아키텍처(Parallel Vision Transformer)
 - 참고: <https://arxiv.org/abs/2302.05442>

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 4 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 3 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 2 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 5 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 6 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 7 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
LOCAL_RANK: 1 - CUDA_VISIBLE_DEVICES: [0,1,2,3,4,5,6,7]
Loading 'train_dataloader' to estimate number of stepping batches.
```

	Name	Type	Params
0	architecture	Architecture	85.8 M
1	architecture.patch_embed	PatchEmbed	741 K
2	architecture.patch_embed.patch_embed	EinMix	590 K
3	architecture.backbone	HomogeneousBackbone	85.0 M
4	architecture.backbone.blocks	ModuleList	85.0 M
5	architecture.pool	Reduce	0
6	loss_fn	SimCLRLoss	0
7	train_loss	MeanMetric	0
8	val_loss	MeanMetric	0

```
Trainable params: 85.8 M
Non-trainable params: 0
Total params: 85.8 M
Total estimated model params size (MB): 171
```

Running script(Docker)

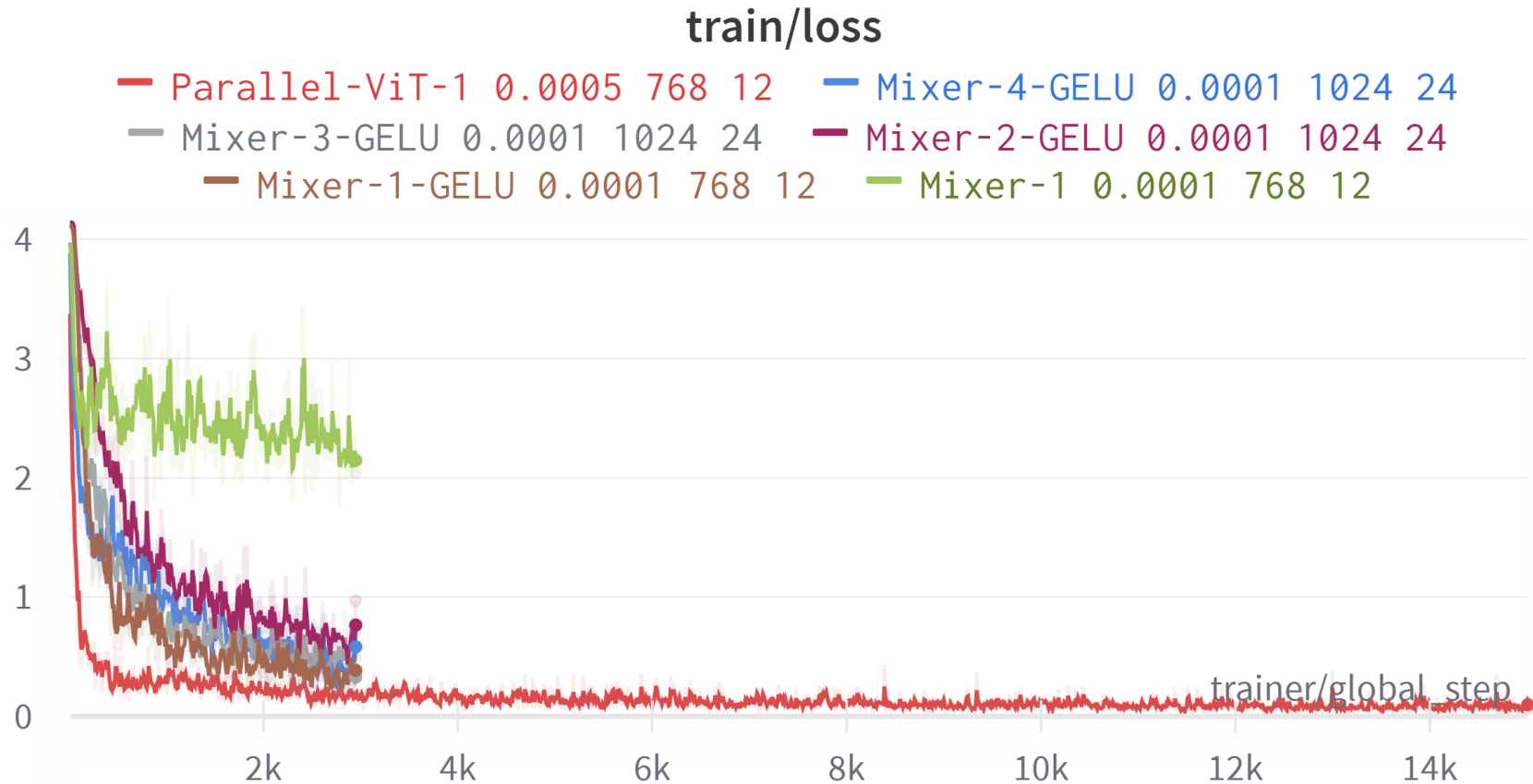
✓ 학습 과정

```
Epoch 0/2 1/5199 0:01:02 • -:--:-- 0.00it/s loss: 3.62 v_num: 0aj6 [2023-09-15 05:16:07,938][torch.nn.parallel.distributed][INFO]
s have been rebuilt in this iteration.
[2023-09-15 05:16:08,881][torch.nn.parallel.distributed][INFO] - Reducer buckets have been rebuilt in this iteration.
[2023-09-15 05:16:08,881][torch.nn.parallel.distributed][INFO] - Reducer buckets have been rebuilt in this iteration.
[2023-09-15 05:16:08,958][torch.nn.parallel.distributed][INFO] - Reducer buckets have been rebuilt in this iteration.
[2023-09-15 05:16:08,959][torch.nn.parallel.distributed][INFO] - Reducer buckets have been rebuilt in this iteration.
[2023-09-15 05:16:08,960][torch.nn.parallel.distributed][INFO] - Reducer buckets have been rebuilt in this iteration.
[2023-09-15 05:16:08,965][torch.nn.parallel.distributed][INFO] - Reducer buckets have been rebuilt in this iteration.
[2023-09-15 05:16:09,038][torch.nn.parallel.distributed][INFO] - Reducer buckets have been rebuilt in this iteration.
Epoch 0/2 46/5199 0:01:43 • 1:16:11 1.13it/s loss: 1.8 v_num: 0aj6
```

Running script

✓ 학습 결과 예시 (Weights & biases)

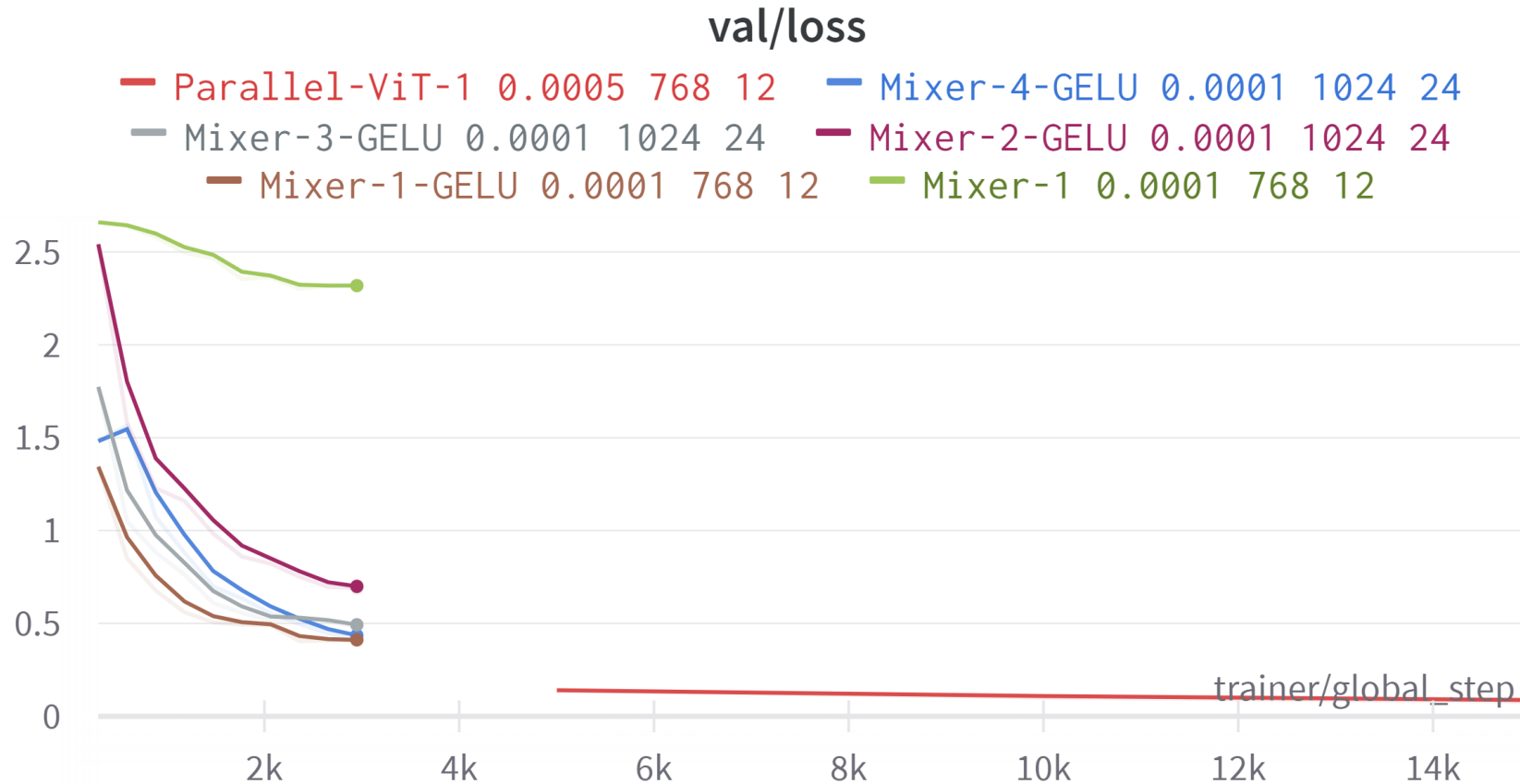
- 참고: <https://wandb.ai/jihoahn9303/groovis/groups/Mixer-exp/workspace>



Running script

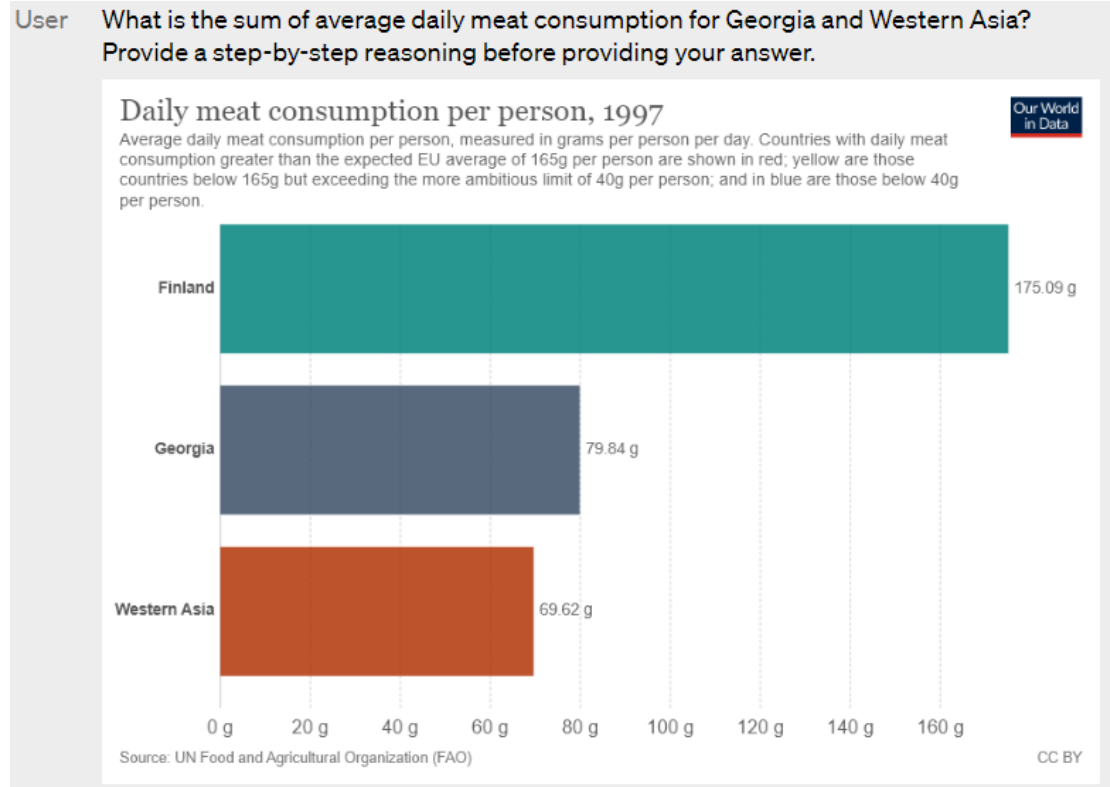
✓ 학습 결과 예시 (Weights & biases)

- 참고: <https://wandb.ai/jihoahn9303/groovis/groups/Mixer-exp/workspace>



Model Usage

- ✓ 모델 학습 결과 사용 예시(GPT-4): 멀티 모달 학습 방식을 이용한 질의 응답
 - 참고: <https://openai.com/research/gpt-4>



GPT-4 To find the sum of average daily meat consumption for Georgia and Western Asia, we will follow these steps:

1. Identify the average daily meat consumption for Georgia.
2. Identify the average daily meat consumption for Western Asia.
3. Add the values from steps 1 and 2.

Step 1: From the provided chart, we see that the average daily meat consumption for Georgia is 79.84 grams per person per day.

Step 2: From the chart, we see that the average daily meat consumption for Western Asia is 69.62 grams per person per day.

Step 3: Add the values from steps 1 and 2.
 $79.84 \text{ g (Georgia)} + 69.62 \text{ g (Western Asia)} = 149.46 \text{ g}$

So, the sum of average daily meat consumption for Georgia and Western Asia is 149.46 grams per person per day.

Model Usage

- ✓ 모델 학습 결과 사용 예시: 트랜스포머를 활용한 주가 예측 (Stock market index prediction using deep Transformer model)
 - 참고: <https://www.sciencedirect.com/science/article/abs/pii/S0957417422013100>

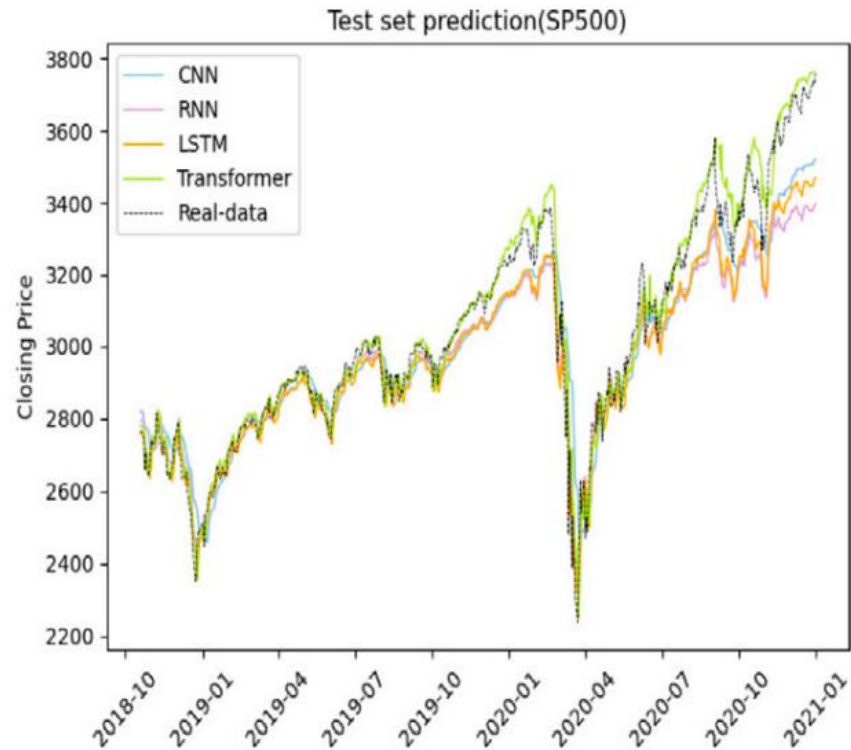


Table 2

The means and standard errors of three indicators across 10 independent experiments for four datasets.

Index	Models	MAE	MSE	MAPE
CSI 300	RNN	0.0825(0.0035)	0.0127(0.0010)	1.2986(0.0503)
	CNN	0.0948(0.0020)	0.0170(0.0006)	1.4816(0.0306)
	LSTM	0.0693(0.0091)	0.0091(0.0002)	1.1216(0.0144)
	Transformer	0.0641(0.0025)	0.0079(0.0004)	0.9549(0.0317)
Nikkei 225	RNN	0.0680(0.0046)	0.0086(0.0004)	1.4477(0.1044)
	CNN	0.0851(0.0036)	0.0158(0.0009)	1.7881(0.0870)
	LSTM	0.0585(0.0025)	0.0066(0.0003)	1.2438(0.0572)
	Transformer	0.0471(0.0017)	0.0043(0.0002)	1.0072(0.0343)
Hang Seng	RNN	0.1719(0.0073)	0.0446(0.0040)	1.9048(0.2489)
	CNN	0.1506(0.0122)	0.0353(0.0059)	1.7989(0.1514)
	LSTM	0.0985(0.0047)	0.0164(0.0013)	1.1623(0.0584)
	Transformer	0.0881(0.0025)	0.0138(0.0005)	1.0403(0.0271)
S&P 500	RNN	0.1359(0.0274)	0.0321(0.0087)	2.2567(0.4682)
	CNN	0.1533(0.0188)	0.0414(0.0101)	2.5158(0.2709)
	LSTM	0.1092(0.0256)	0.0236(0.0099)	1.7768(0.4001)
	Transformer	0.0814(0.0131)	0.0145(0.0037)	1.3800(0.2163)

Reference

Contrastive learning: Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020, November). A simple framework for contrastive learning of visual representations. In *International conference on machine learning* (pp. 1597-1607). PMLR. <https://nips.cc/media/neurips-2021/Slides/21895.pdf>

Residual Connection: He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Layer normalization: Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

MLP-Mixer: Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., ... & Dosovitskiy, A. (2021). Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34, 24261-24272.

Attention:

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[2] Clark, K., Khandelwal, U., Levy, O., & Manning, C. D. (2019). What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*.

Vision Transformer:

[1] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

[2] Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., ... & Houlsby, N. (2023, July). Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning* (pp. 7480-7512). PMLR.

[3] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 10012-10022).

Positional Embedding: Wang, Y. A., & Chen, Y. N. (2020). What do position embeddings learn? an empirical study of pre-trained language model positional encoding. *arXiv preprint arXiv:2010.04903*.

Transformer in real world: Wang, C., Chen, Y., Zhang, S., & Zhang, Q. (2022). Stock market index prediction using deep Transformer model. *Expert Systems with Applications*, 208, 118128.

Thank you