

Cogsys Winter Code Hack: 4회차

End-to-end pipeline with collaboration



Daehyun Cho

Cognitive System Lab

A.I. Dept, Korea Univ.

전체적인 계획

1. 대회 참여하기 & GitHub Repository 개설하기D
2. Pytorch DataLoader 구축하기 & Code Review하기
3. Pytorch Model 개발하기 & 제출해보기
4. 여러 가지 실험을 하기 위해 세팅하기 (w/ Hydra) & wandb logging하기
5. Pytorch-Lightning으로 갈아타보기
6. Accuracy 100% 만들기

이 과정을 조금 불친절하게 진행할 예정

지향

- ✓ 알잘딱깔센
- ✓ Collaborations
- ✓ 조금이라도 모르면 구글링 (선무당이 사람잡음)

지양

- ✗ Copy & Paste: 단순한 syntax정도는 괜찮은데, 전체 파이프라인을 베껴오는 건 취지를 아주 많이 벗어남
- ✗ 나의 온 시간을 투자하기: 짬짬히... 틈틈히 진행하는 것을 권장 (교수님한테 혼나기 싫음)

TODO List

- ✓ Instantiate class objects with hydra
 - ✓ Create your own configuration yaml file
 - ✓ Substitute instantiation as much as you can, with hydra
 - ✓ Run train codes with CLI, changing hyperparameters
-
- ✓ Make wandb account via authorizing github account
 - ✓ Run sample wandb code and understand the logics behind
 - ✓ Log EVERYTHING with wandb

During todos...

- ✓ Commit middle works as much as you can (make it unconscious)
- ✓ Push your work to remote repository



- ☒ 지난 시간에 만든 training code를 돌려보세요
- ☒ batch size를 바꿔서 훈련 시키고 싶으면 어떻게 해야하나요?



```
import timm
from digitrec.trainer import Trainer

if __name__=="__main__":
    model = timm.create_model("resnet10t", num_classes=10,
in_chans=1)
    trainer = Trainer(model=model, optimizer="adam")

    trainer.fit(epochs=10)
```



- Batch size 바꾸려면 스크립트에 들어가서 코드를 수정해야하는 것을 알 수 있습니다.
- 우리가 훈련에 필요한 여러 변수들을 하나의 dictionary 안에 넣어 놓고 관리하면 훨씬 다양한 실험을 할 수 있습니다.
 - 원래 창의력은 어느 정도 제반이 갖춰져야 떠오릅니다. 현실적으로 당장 눈앞에 있는 것도 실행이 어려우면 그 뒤를 생각하기 어렵거든요
- 이를 쉽게 돕기 위해 hydra라는 라이브러리가 있습니다.
- 그 전에 우리가 github workflow를 구성하기 위해 생성했던 main.yml 파일을 기억하시나요?
 - Yaml은 key-value 형태의 데이터를 뛰어난 가독성으로 보관할 수 있는 파일입니다.

```
batch_size: 16
model:
  depth: 34
  in_planes: [48, 64, 96]
  no_max_pool: False
```



Managing Configurations

- Batch size 바꾸려면 스크립트에 들어가서 코드를 수정해야하는 것을 알 수 있습니다.
- 우리가 훈련에 필요한 여러 변수들을 하나의 dictionary 안에 넣어 놓고 관리하면 훨씬 다양한 실험을 할 수 있습니다.
 - 원래 창의력은 어느 정도 제반이 갖춰져야 떠오릅니다.
 - 현실적으로 당장 눈앞에 있는 것도 실행이 어려우면 그 뒤를 생각하기 어렵습니다.
- 이를 쉽게 돕기 위해 hydra라는 라이브러리가 있습니다.
 - Hydra는 여러 개의 yaml 파일을 관리할 수 있게 도와줍니다.
 - github workflow를 구성하기 위해 생성했던 main.yml 파일을 기억하시나요?
 - Yaml은 key-value 형태의 데이터를 뛰어난 가독성으로 보관할 수 있는 파일입니다.

```
batch_size: 16
model:
  depth: 34
  in_planes: [48, 64, 96]
  no_max_pool: False
```



Managing Configurations

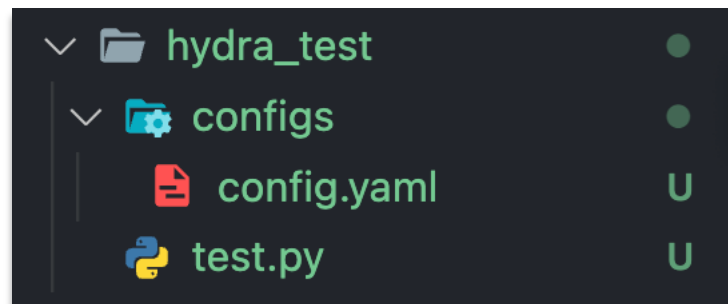
- 단순 copy & paste를 지양하기 위해 여기에 예제 코드를 많이 안 넣는 편인데, hydra의 경우 예시 몇 개를 돌려보면 구조를 파악하기 쉽기 때문에 아래와 같은 예제를 만들어주세요

```
import hydra
from omegaconf import DictConfig, OmegaConf

@hydra.main(version_base=None, config_path=".", config_name="config")
def test(cfg: DictConfig) → None:
    breakpoint()
    print(OmegaConf.to_yaml(cfg))

if __name__ == "__main__":
    test()
```

```
batch_size: 16
model:
  depth: 34
  in_planes: [48, 64, 96]
  no_max_pool: False
```





Managing Configurations

- breakpoint는 python=3.6부터 도입된 python debugging 명령어입니다.
- breakpoint()를 걸어두면 스크립트를 실행시킬 때 해당 위치에서 멈추고, 그 안의 변수들을 조회할 수 있습니다.
- ☒ 방금 만든 스크립트를 실행시키고 breakpoint에서 멈추면, cfg 변수를 살펴보세요. 안의 변수들을 어떻게 조회할 수 있는지 살펴봅시다.
- ☒ 저기서 model의 depth를 바꾸고 싶으면 어떻게 실행시켜야할까요?

```
~/codespace/digit-recognizer/hydra_test main*  
digit > python test.py  
> /Users/daehyuncho/codespace/digit-recognizer/hydra_test/test.py(8)test()  
-> print(OmegaConf.to_yaml(cfg))  
(Pdb) cfg  
{'batch_size': 16, 'model': {'depth': 34, 'in_planes': [48, 64, 96], 'no_max_pool': False}}  
(Pdb) cfg.model.depth  
34  
(Pdb) cfg.model  
{'depth': 34, 'in_planes': [48, 64, 96], 'no_max_pool': False}  
(Pdb) █
```



Managing Configurations

- 아주 쉽게 CLI에서 model.depth=50 을 추가해주시면 됩니다.
- 벌써부터 어떻게 응용할 수 있을지 많이 보이네요.

```
~/codespace/digit-recognizer/hydra_test main*
digit > python test.py model.depth=50
> /Users/daehyuncho/codespace/digit-recognizer/hydra_test/test.py(8)test()
-> print(OmegaConf.to_yaml(cfg))
(Pdb) cfg
{'batch_size': 16, 'model': {'depth': 50, 'in_planes': [48, 64, 96], 'no_max_pool': False}}
(Pdb) cfg.model.depth
50
(Pdb) █
```



- 이번에는 hydra를 이용해 객체를 생성해봅시다.
 - Hydra는 dict 내에 `_target_`이라는 변수를 가지고 있으면 `instantiate`이라는 메소드를 통해 객체화를 진행시킬 수 있습니다.
- 가령 `MSELoss`를 만들어주고 싶으면 아래와 같이 생성할 수 있습니다.

```
import hydra
from omegaconf import DictConfig, OmegaConf

@hydra.main(version_base=None, config_path="configs", config_name="config")
def test(cfg: DictConfig) → None:
    loss_fn = hydra.utils.instantiate(cfg.loss)
    breakpoint()
    print(OmegaConf.to_yaml(cfg))

if __name__ == "__main__":
    test()
```

```
batch_size: 16
model:
  depth: 34
  in_planes: [48, 64, 96]
  no_max_pool: False

loss:
  _target_: torch.nn.MSELoss

defaults:
  - _self_
  - optimizer: base
```



Managing Configurations

- 너무 쉽게 생성이 된 걸 볼 수 있습니다.
- 공장 객체화를 해주는 게 중요한 이유는 여러 가지 지저분한 상황을 방지할 수 있습니다.

```
~/codespace/digit-recognizer/hydra_test main* ↑  
digit > python test.py  
> /Users/daehyuncho/codespace/digit-recognizer/hydra_test/test.py(9)test()  
-> print(OmegaConf.to_yaml(cfg))  
(Pdb) loss_fn  
MSELoss()  
(Pdb) █
```

MSELOSS

CLASS `torch.nn.MSELoss(size_average=None, reduce=None, reduction='mean')` [\[SOURCE\]](#)



Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y .

The unreduced (i.e. with `reduction` set to `'none'`) loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2,$$

where N is the batch size. If `reduction` is not `'none'` (default `'mean'`), then:



- 공장 객체화를 해주는 게 중요한 이유는 여러 가지 지저분한 상황을 방지할 수 있습니다.
- 오른쪽처럼 `get_loss` 함수를 짰다고 가정합시다.
 1. 만약 str에 대문자 MSE가 들어왔다면?
 2. 만약 MSELoss에서 reduction="mean"이 필요하다면?
 3. 만약 Cross-entropy loss가 필요하다면?
- 매번 코드를 추가해줘야 하는 부담스러움 & 귀찮음이 있습니다.
 - 아니 그러면 매번 짜면 되지 않냐? 코드를 많이 짜는 게 중요하긴한데, 불필요한 단순작업 코드가 많을 수록 실수할 확률이 높아집니다.
 - 무엇보다 hydra 자체가 단순해서 부하가 그렇게 크지 않습니다.
- 하지만 hydra를 사용하면 위에서 제기한 문제들은 **쉽게 해결**이 가능합니다.
 - 1번의 경우 처음부터 `_target_: torch.nn.MSELoss`로 잘 넣어주기 때문에 이슈가 없음
 - 2번의 경우 `_target_`과 같은 레벨에 `reduction: mean`을 넣어주면 됩니다
 - 3번의 경우 `_target_: torch.nn.CrossEntropyLoss`를 넣어주면 됩니다.
- 굉장히 쉽게 많은 것을 해결할 수 있습니다.



```
import torch

def get_loss(loss_fn: str):
    if loss_fn == "mse":
        loss_fn = torch.nn.MSELoss()
    elif loss_fn == "l1":
        loss_fn = torch.nn.L1Loss()
    return loss_fn
```



Train models

- ☒ 이제 여러분의 훈련 프로세스에 hydra를 넣어서 깔끔하게 만들어보세요
- 너무 힘들면 해당 커밋을 확인해보세요
 - <https://github.com/Cognitive-Systems-Laboratory/digit-recognizer/commit/ced680061e42034ba5a09203b344572d622b6d82>

```
~/codespace/digit-recognizer main* ↑  
● digit > python main.py
```



What is happening

Logging

- 현재까지 짠 코드에는 logging이 전혀 없습니다. 그래서 훈련을 돌려도 아래처럼 아무 것도 나오지 않아요

```
~/codespace/digit-recognizer main*  
● digit > python main.py
```

- 훈련을 돌리는 것도 중요한데, 훈련이 뭐하고 있는지 아는 것도 중요합니다. 최대한 많은 값들을 되는데로 찍는 게 좋습니다.
- Logging을 하는 방법은 여러 가지가 있습니다.
 - Python built-in `logging` library
 - Third-party libraries: wandb, mlflow, Neptune, aimstack
- 오늘은 **wandb**만 다룰 예정인데, built-in library인 **logging**도 무조건 알아야합니다 (앞으로 print 대신에 logging을 쓰도록 하세요)
 - <https://docs.python.org/3/library/logging.html>
 - <https://ooog.tistory.com/9>



- Wandb 홈페이지에 가서 계정부터 만드세요. Github 계정으로 연동이 됩니다.
- Github 계정으로 연동되면 좋은 점이 해당 experimen가 어떤 commit에서 실행되었는지 알 수 있습니다.

Weights & Biases Platform Solutions Ecosystem Docs Pricing Enterprise Company

SIGN IN SIGN UP

The developer-first MLOps platform

Weights & Biases makes it easy to track your experiments, manage & version your data, and collaborate with your team so you can focus on building the best models.

SIGN UP REQUEST DEMO

Loved by 500,000+ ML practitioners

Sweep: shape_sweep

Name (156 visualized)	val-full/a
lucky-sweep-5	0.981
northern-sweep-111	0.9804
atomic-sweep-63	0.9795
drawn-sweep-169	0.9793
proud-sweep-42	0.9784
visionary-sweep-118	0.9781
scarlet-sweep-79	0.976
rural-sweep-109	0.9756
scarlet-sweep-62	0.9756
super-sweep-23	0.9754
morning-sweep-139	0.9754
prime-sweep-135	0.9752
fresh-sweep-86	0.9752
rose-sweep-43	0.9751
rose-sweep-57	0.9748
easy-sweep-1	0.9748

Results of Hyperparameter Sweep 5

Parameter importance with respect to val-full/accuracy

Config parameter	Importance	Correlation
n_params	High	High
budget	Medium	High
parameter_budget	Medium	High
Runtime	Low	Low
lr	Low	High

Best Model: lucky-sweep-5, val-full/accuracy: 0.981



Logging

- 기본적으로 계층구조는 Project - experiment입니다
 - MNIST를 학습하는 실험들을 모아두는 공간 = project
 - 여기서는 ann-2022-assignment2
 - 학습한 실험 한 개 = run
 - Runs
- <https://wandb.ai/1pha/ann-2022-assignment2> 예제참고

1pha > Projects > ann-2022-assignment2 > Table

Runs (93)

<



- 가장 기초적인 사용법은 홈페이지에 있어요
 - wandb.init: 어떤 프로젝트에 어떤 run을 넣을 것인지
 - Config를 wandb에 넣어줍니다.
 - Wandb.watch(model)의 경우는 gradient 분포를 자동으로 기록해줍니다. 귀찮으면 안해도 됩니다.
 - wandb.log({"loss": loss}) 처럼 어떤 key-value 형태의 dict를 기록합니다.
- Wandb 홈페이지만 로그인하는 게 아니라 CLI에서도 로그인해야합니다.
- Command line에서 `wandb login` 을 치고 API-key를 넣습니다.
 - API key는 wandb 홈페이지 Setting에 가면 발급하여 복사해서 넣으세요

```
import wandb
```

```
# 1. Start a new run
```

```
wandb.init(project="gpt-3")
```

```
# 2. Save model inputs and hyperparameters
```

```
config = wandb.config
```

```
config.learning_rate = 0.01
```

```
# 3. Log gradients and model parameters
```

```
wandb.watch(model)
```

```
for batch_idx, (data, target) in
```

```
enumerate(train_loader):
```


```
    if batch_idx % args.log_interval == 0:
```

```
# 4. Log metrics to visualize performance
```

```
wandb.log({"loss": loss})
```



Logging

-  기본 예제 말고 우리도 써봅시다.
- 기존에 있던 main 함수에 wandb만 넣어서 우선 돌려보고 확인해봅시다.

```
import numpy as np
import hydra
import omegaconf
import wandb

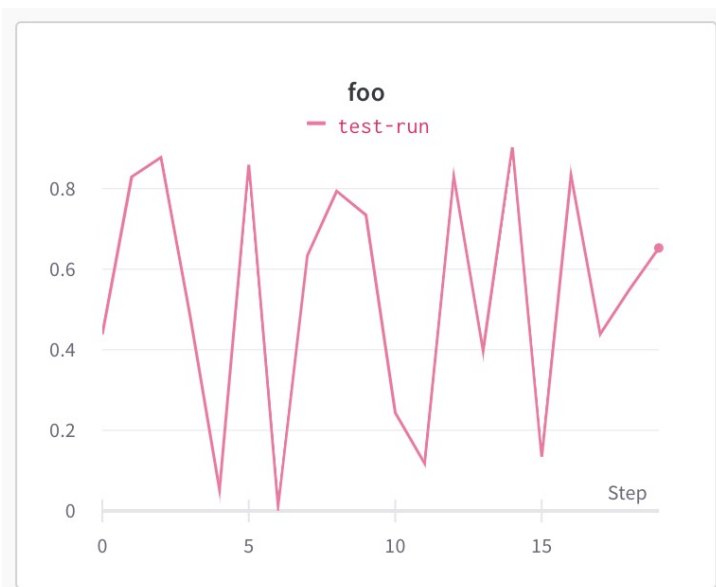
@hydra.main(version_base="1.3", config_path="configs", config_name="train")
def main(cfg: omegaconf.DictConfig) → None:
    wandb.init(
        project="test-project",
        name="test-run",
        config=omegaconf.OmegaConf.to_container(cfg, resolve=True, throw_on_missing=True),
    )
    for i in np.random.random(size=20):
        wandb.log({"foo": i})

if __name__ == "__main__":
    main()
```



Logging

- 방금 돌린 experiment 페이지에 가서 자세히 살펴보세요.
- ☒ 어떤 게 기록되었나요? 자세히 살펴보세요. 생각보다 많은 게 들어있어요
 - ☒ 어떤 commit에서 진행되었나요?
 - ☒ 어떤 명령어를 통해 해당 experiment가 실행되었나요?



Config

Raw

Config parameters describe your model's inputs. [Learn more](#)

Search keys

Key	Value
data	
target	"torch.utils.data.DataLoader"
batch_size	64
dataset	
target	"digitrec.dataloader.DigitDataset"
data_dir	"./data"
file_name	"train.csv"
device	"mps"
logging	
name	"test-run"
project	"test-project"
model	
in_chans	1
model_name	"resnet10t"
num_classes	10
modes (2 collapsed)	
optimizer	



Logging

- Configuration을 왜 저렇게 넣나요? Hydra가 dict 구조이긴한데, 정확히는 dict를 상속받은 omegaconf라는 다른 객체라서 그렇습니다.
- 예를 그냥 dict(cfg)로 파싱해서 넣으면 이상하게 나와요. Wandb에서 방법을 제시해줬습니다.
- <https://wandb.ai/adrishd/hydra-example/reports/Configuring-W-B-Projects-with-Hydra--VmIldzoxNTA2MzQw>

🔗 ▼ Pitfall #1: Using Hydra's Config with wandb.config

Hydra uses `omegaconf` as the default way to interface with the configuration dictionaries. However, it is very important to keep in mind that OmegaConf's dictionary is *not* a subclass of primitive dictionaries (unlike tools like `Addict`). Hence directly passing Hydra's Config to `wandb.config` leads to unexpected results on the dashboard. It's necessary to convert `omegaconf.DictConfig` to primitive `dict()` type, before passing to `wandb.config`.

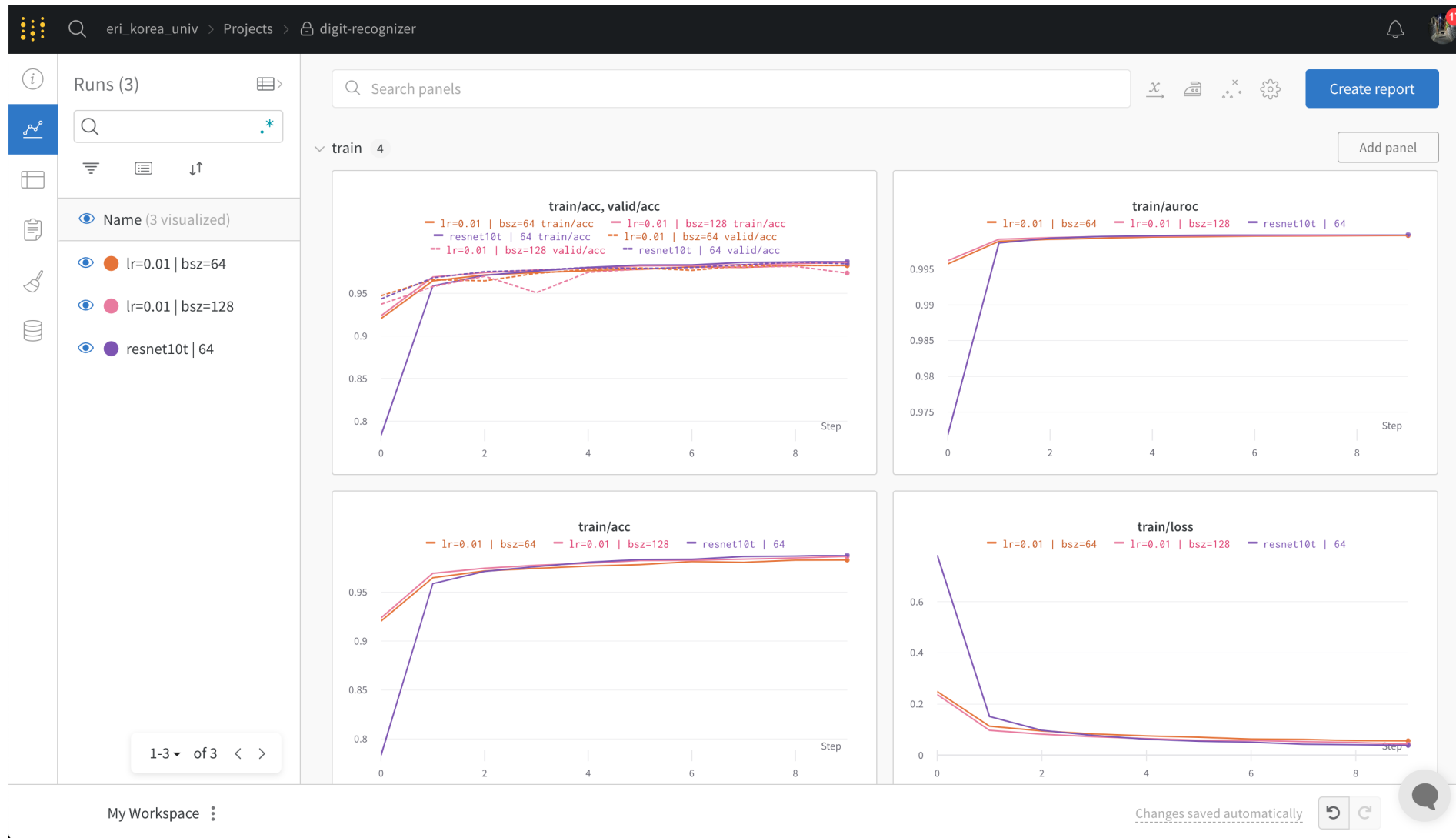
```
@hydra.main(config_path="configs/", config_name="defaults")
def run_experiment(cfg):
    wandb.config = omegaconf.OmegaConf.to_container(
        cfg, resolve=True, throw_on_missing=True
    )
    run = wandb.init(entity=cfg.wandb.entity, project=cfg.wandb.project)
    wandb.log({"loss": loss})
    model = Model(**wandb.config.model.configs)
```



- Wandb를 이용해서 우리 학습기록을 싹 다 넣어봅시다.
 - ☒ wandb project와 run-name을 hydra로 조절해봅시다.
 - ☒ **torchmetrics**라는 라이브러리를 통해 Accuracy / AUROC 같은 다른 metric도 계산해서 넣어봅시다.
 - 과거에는 직접 계산하거나 sklearn을 이용했는데, torchmetrics가 훨씬 편합니다...
 - ☒ 중간 중간에 이미지도 로깅해봅시다. 참고: <https://docs.wandb.ai/guides/track/log/media> (예제코드에 안 넣었어요. 한 번 해보세요)
 - ☒ 예제코드에는 tqdm도 같이 넣어놨습니다. 한 번 확인해보세요.
 - ☒ wandb 내에 있는 plot들은 customize할 수 있습니다. Train과 validation metric을 하나의 그래프에 그려보세요 (use "Add panel")
 - ☒ 여러 configuration들을 바꿔가면서 돌리고 비교해봅시다.
-
- 평소대로라면 wandb project link를 그냥 드리는데, 로그인인 이상하게 되어서 Public 공개가 안되네요. 이미지로 대체해요



- 평소대로라면 wandb project link를 그냥 드리는데, 로그인이 이상하게 되어서 Public 공개가 안되네요. 이미지로 대체해요



Thank you 🙏

Daehyun Cho

1phantasmas@korea.ac.kr