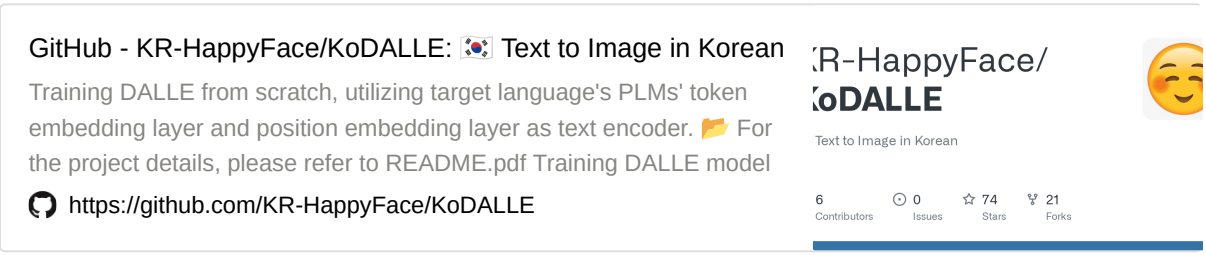
KoDALLE Code



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/0f592fa3-5aec-467f-affc-a67a3fbfffd6/KoDALLE_KEANU.zip

Dependency

```
$ conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
$ conda install tqdm pandas scikit-learn
$ pip install axial_positional_embedding optuna psycopg2-binary pre-commit\
 split-folders ptflops adamp easydict cpython einops
```

문제

```
pip install dalle_pytorch
```

이 명령어를 입력하기 전에 아래 패키지를 다음과 같이 설치하지 않는다면 계속 오류가 난다. (youtokentome를 그냥 pip나 conda로 설치 시 설치안됨)

```
conda install -c conda-forge youtokentome
```

train.py

- WandB 관련 코드 주석처리
- checkpoint 및 데이터 github에 포함 안되어 있음
  - checkpoint 불러오기는 일단 제거
  - 데이터 AI Hub에서 가져와서 포맷 맞추기(ms coco)

Dataset class 수정

- MS Coco
  - korean caption : 하나의 json 파일 내에 모든 이미지의 caption이 저장되어 있음
  - train/val 2014에 대한 caption
- 다른 데이터
  - 목표를 art generation으로 한정하였기 때문에 해당 data를 구해와야 한다.

klue/roberta checkpoint 불러오기

- 현재 코드에서는 weight를 `roberta_large_wte.pt` 및 `roberta_large_wpe.pt`로 불러옴(각각 token embedding weight 및 position embedding weight)
- 파일 확장자가 `.pt`인 것을 봐서는 huggingface에서 모델을 불러와서 torch.save로 저장한 것을 불러온 것으로 보임.

```
from transformers import AutoModel, AutoTokenizer
import torch

model = AutoModel.from_pretrained("klue/roberta-large")

print(model)

torch.save(model.embeddings.position_embeddings, './Roberta/roberta_large_wpe.pt')
torch.save(model.embeddings.word_embeddings, './Roberta/roberta_large_wte.pt')
```

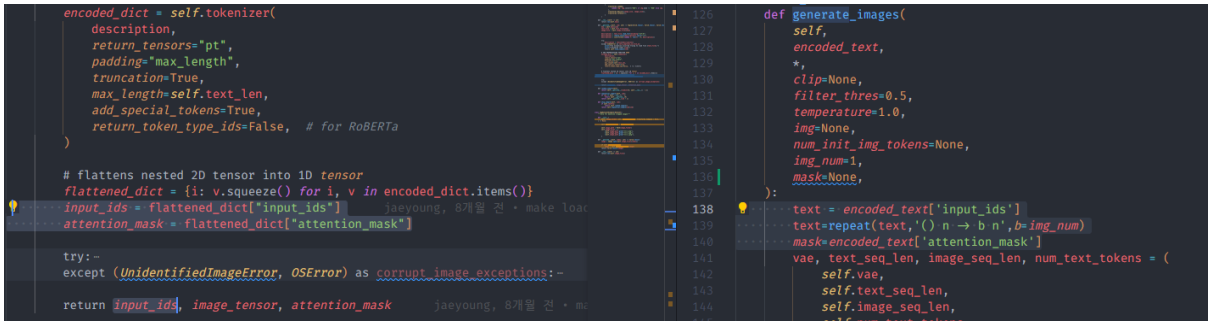
“attention type "f" is not valid”

- dalle_torch 내부 `transformer` class의 __init__에서 attn_type 오류(attn_type="full")
- `attn_type` 파라미터를 `None`으로 변경 시 tranformer 내부에서 `attn_type=("full",)`로 변경
- attn_type를 slice하는 코드에서 ('full', ...)이 아닌 ('f', 'u', 'l', 'l') 형태로 들어가서 오류가 생긴 것으로 추정됨

IndexError: too many indices for tensor of dimension 2

```
File "c:\Users\PC\Downloads\SKT AI\KoDALLE\dalle\models.py", line 138, in generate_images
    text = encoded_text['input_ids']
IndexError: too many indices for tensor of dimension 2
```

- Baseline 코드의 완성도가 크게 의심되는 오류.
- 요약하자면 Dataset에서 dictionary를 그대로 안 넘겨주고 필요한 부분만 tuple로 넘겨주었는데 model의 `generate_images`함수에서는 dictionary로 넘긴것으로 취급함. `torch.tensor`에 key indexing을 하고 있음.

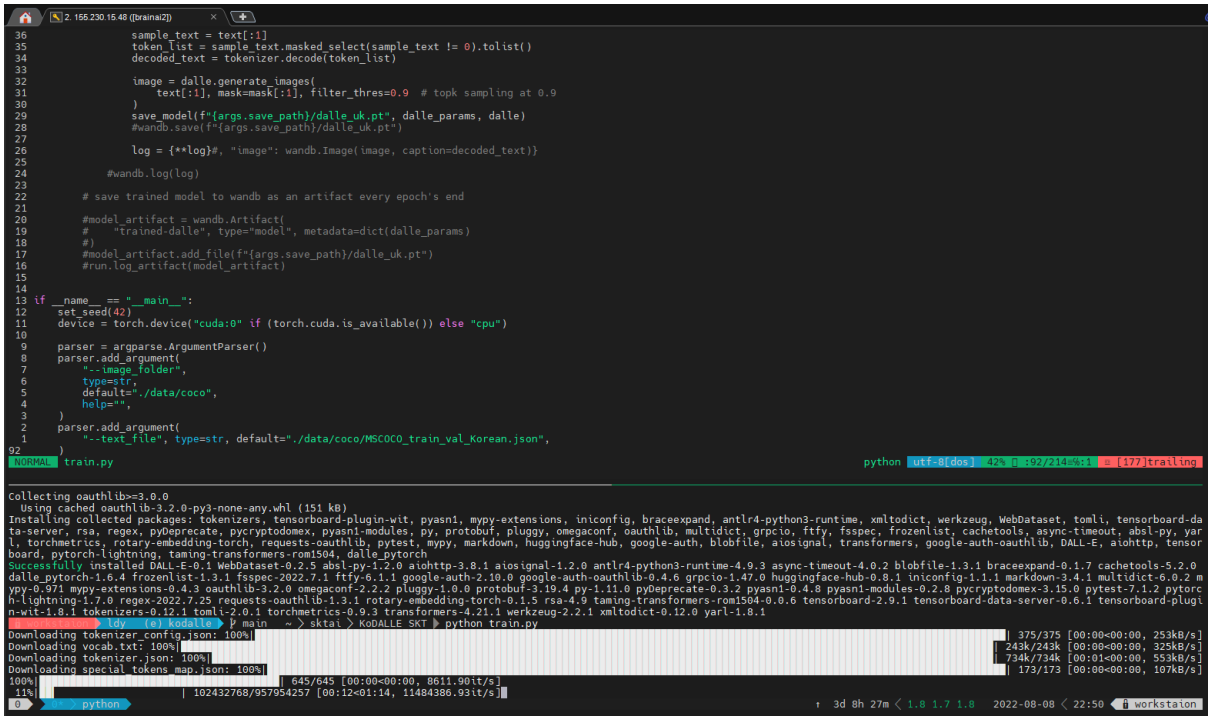


- `generate_images` 함수에서 다음과 같이 변경함

```
text = encoded_text['input_ids']
mask=encoded_text['attention_mask']
```

```
text = encoded_text#['input_ids']
# mask=encoded_text['attention_mask']
```

코드동작 확인 완료



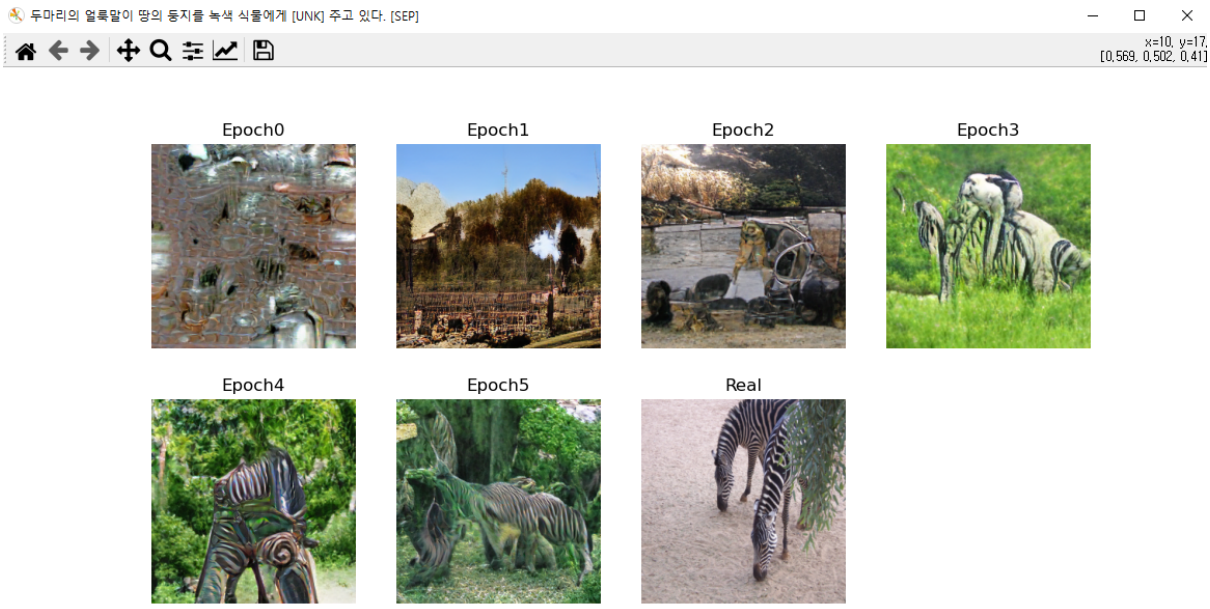
Inference 코드 작성

Training

train config 1

Data	MS COCO w/ 한국어 caption
Text Encoder	KLUE/RoBERTa
VQGAN	원 논문 VQGAN
VQGAN Token Decoding Tranformer	16 layer
Device	RTX 3090ti x 1
Batch size	24
Epoch	5
Date	2022-08-09

Sampled image



Analysis

1. Data and results
    - 훈련에 문제가 없음을 확인
    - MS COCO w/ 한국어 caption에서 caption 부분이 어색하게 해석된 부분 꽤 있음
    - 현재 loss는 5 epoch 동안 5.3 → 4.4 정도에서 수렴됨(epoch 3~5까지는 큰 차이 없는듯)
    - RTX3050에서 이미지 하나(256*256) 당 inference time은 대략 10초 정도이다.
2. Code

- 한 이미지당 여러 caption이 있는데 그중 랜덤하게 하나를 뽑아서 훈련(computational resource limit 때문인듯, augmentation 효과?)

---

train config 2 Plans

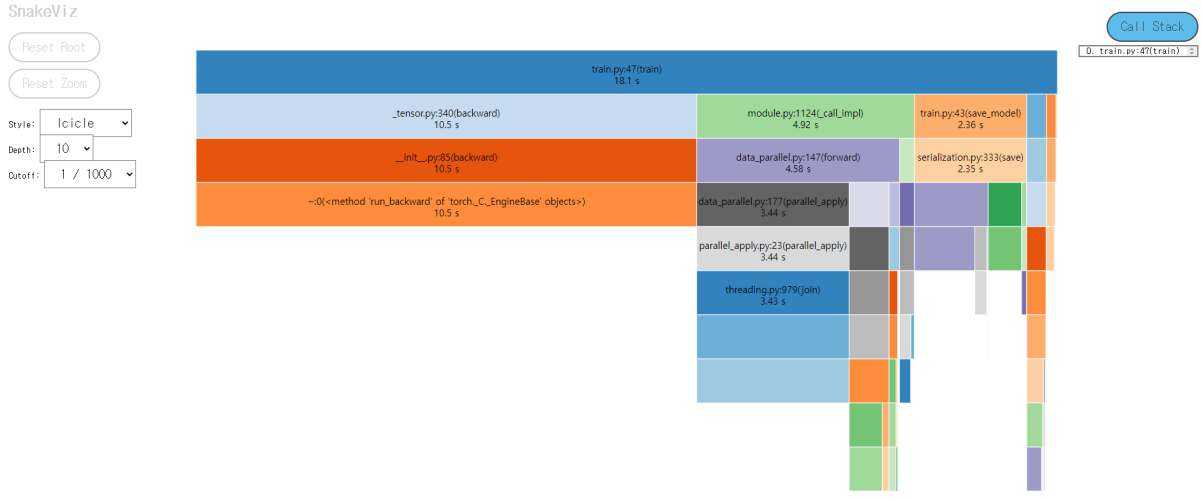
Data	**MS COCO w/ 한국어 caption + other data**
Text Encoder	KLUE/RoBERTa
VQGAN	원 논문 VQGAN
VQGAN Token Decoding Tranformer	**32 layer (?)**
Device	**RTX 3090ti x 2~3**

Batch size	24
Epoch	5
Planed Date	2022-08-15

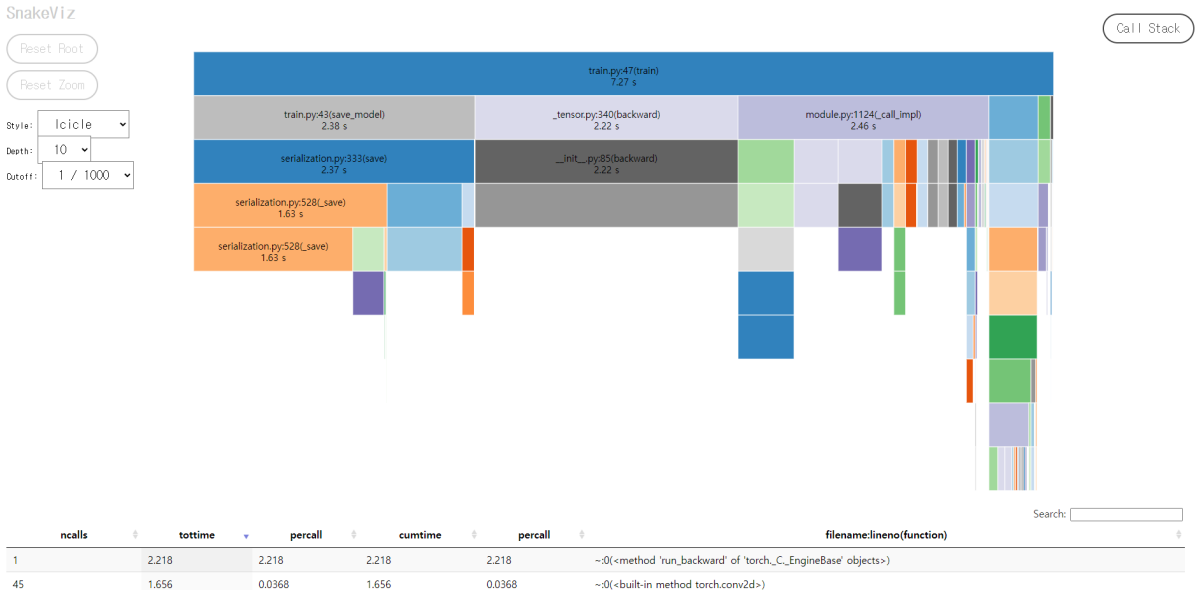
Issue with DataParallel

GPU 2장에 대해 `nn.Dataparallel` 사용 시 1 epoch당 걸리는 시간이 2시간에서 12시간으로 늘어남(batch_size를 GPU 개수에 비례하여 늘림에도 불구하고)

Time profile Analysis



[With DataParallel]



[Without Dataparallel]

`nn.Dataparallel` 사용 시 `loss.backward()`에 소요되는 시간이 2초에서 10초로 대폭 늘어나 증가된 시간에서 큰 비중을 차지하는 모습을 볼 수 있다. `backward()`의 내부는 c++로 구현되어 있어서 그런지 이 이상으로는 내부가 profile되지 않아 정확한 원인을 파악하기 어렵다.

추측되는 바로는 여러 GPU에 걸쳐 계산된 loss가 backpropagation 시 GPU사이 Synchronization으로 인한 overhead가 발생하여 생기는 것 같으며, 모델구조가 단순한 feed forward가 아닌 점 또한 관련있어 보인다.

Backward 과정을 Profiling할 수 있는 방법이 필요해 보인다.

Experiments

1. 일단 `backward`에서 시간소요가 증가한 것이 전체시간 증가에 가장 큰 영향을 미치고 있으므로, gradient update의 대상이 되는 transformer decoder의 parameter의 `requires_grad`를 False로 한 뒤 GPU 2장에 대하여 다시 train을 하였을 시 시간이 얼마나 줄어드는지 확인하는 실험을 진행하였다.

```
class DALLE_Klue_Roberta(nn.Module):
  def __init__(...):
    ...
    for param in self.transformer.parameters():
      param.requires_grad=False
    ...
```

[not applied]

[applied]

값이 크게 줄어드는 모습을 확인할 수 있었다.
흥미로운 점은 single GPU에 적용시 차이가 거의 없다는 점이다. 이로 미루어 보았을 시 DALLE_Klue_Roberta 내에 존재하는 다른 parameter 중 require_grad가 transformer block 이전에 존재하여 backwards pass 연산량에 영향을 주지 않는다고 추측된다.

[not applied in single GPU]

[applied in single GPU]

2. 다음으로는 transformer decoder를 제외한 나머지 모델들은 freeze되어 있으며, requires_grad가 있는 parameter가 없지만, 혹시나 이 모델들에서 grad 정보들이 backpropagation을 통해 업데이트 되고 GPU 사이 synchronize가 불필요하게 되고 있을 수 있다는 추측을 통해 model.forward()에서 transformer decoder를 사용하는 부분의 앞쪽을 torch.no_grad()를 통해 어떠한 grad operation도 일어나지 않도록 하고 train을 하는 실험을 하였다.

```
class DALLE_Klue_Roberta(nn.Module):
  def forward(...):
    with torch.no_grad():
      ...
    out = self.transformer(token)
    ...
```

```
0.25064826011657715 s
epoch : 0:   1%|                                                  | 13/2568 [00:25<1:03:20,  1.49s/it]
0.25281357765197754 s
epoch : 0:   1%|                                                  | 14/2568 [00:26<1:03:09,  1.48s/it]
0.2512078285217285 s
epoch : 0:   1%|                                                  | 15/2568 [00:28<1:02:52,  1.48s/it]
```

[applied in 2 GPUs]

```
0.0029146671295166016 s
epoch : 0:   0%|                                                  | 17/5136 [00:17<1:06:51,  1.28it/s]
0.002962827682495117 s
epoch : 0:   0%|                                                  | 18/5136 [00:18<1:06:50,  1.28it/s]
0.0029249191284179688 s
epoch : 0:   0%|                                                  | 19/5136 [00:18<1:07:00,  1.27it/s]
```

[applied in single GPU]

두 경우 모두에 대해서 연산속도가 훨씬 증가하였으나, 여전히 Multi-GPU 환경에서보다 single GPU일 때 더욱 빠른 연산속도를 보였다.