🤗 Transformers

- 다양한 기능들, 그리고 뇌피셜 살짝 -

2020.06.03 박장원

사용법은 다들 저보다 더 잘 아시지 않을까...

이걸로 어디까지 응용할 수 있는지 알려드릴 예정!

1. 변천사

Huggingface Transformers

pytorch-pretrained-bert

- BERT 논문 출시 1개월 후
- Pytorch로 짠 BERT

pytorch-transformers

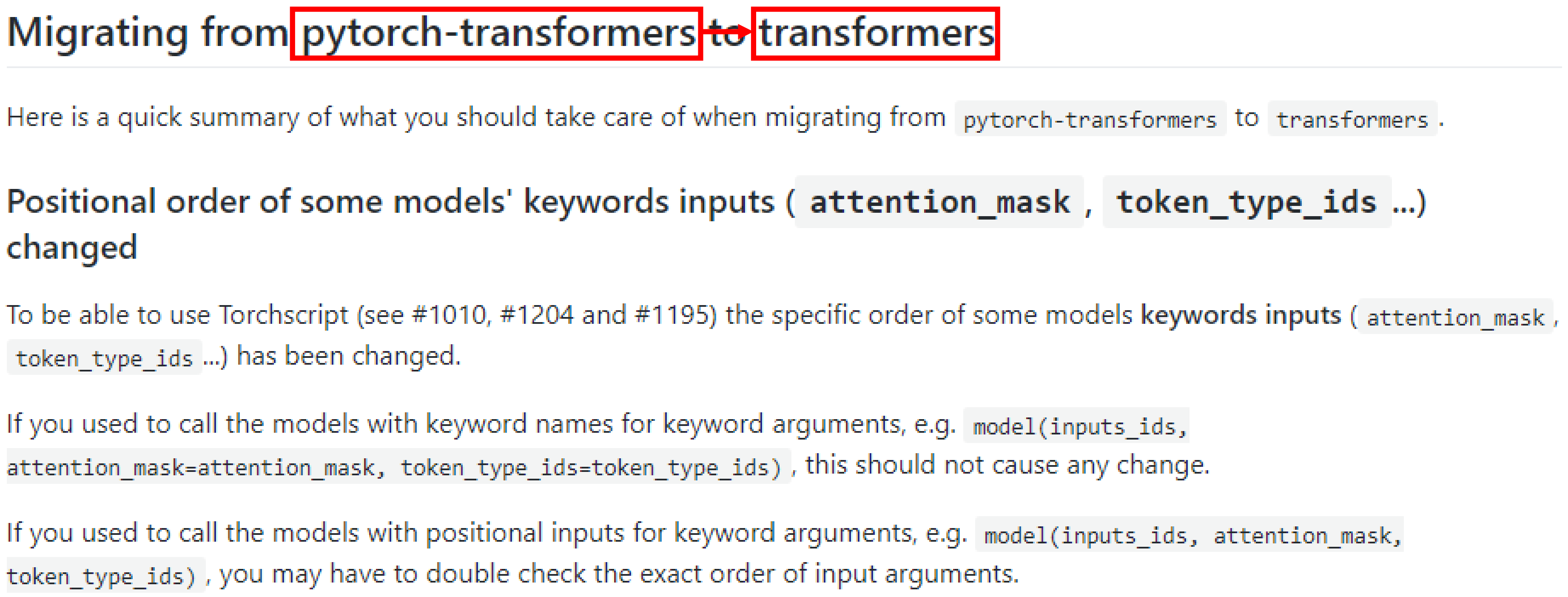
- BERT 이외의 XLNET 등등 함께 지원

transformers

- Tensorflow 2.0도 지원

-> 위의 2개의 라이브러리를 쓴 옛 오픈소스도 있으니 참고!

그대로 가져다 쓰면....



Migrating from pytorch-transformers → transformers

Here is a quick summary of what you should take care of when migrating from `pytorch-transformers` to `transformers`.

Positional order of some models' keywords inputs (`attention_mask` , `token_type_ids` ...) changed

To be able to use Torchscript (see #1010, #1204 and #1195) the specific order of some models **keywords inputs** (`attention_mask` , `token_type_ids` ...) has been changed.

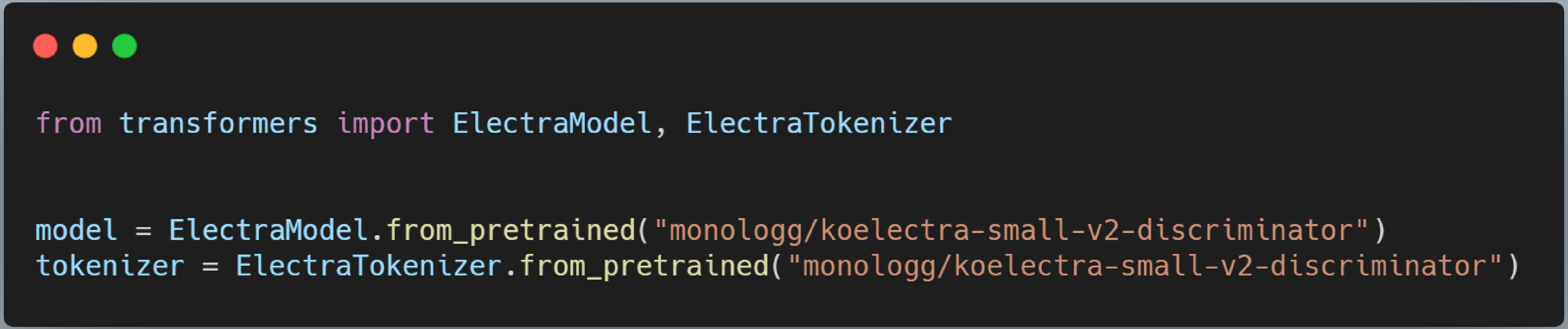
If you used to call the models with keyword names for keyword arguments, e.g. `model(inputs_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)` , this should not cause any change.

If you used to call the models with positional inputs for keyword arguments, e.g. `model(inputs_ids, attention_mask, token_type_ids)` , you may have to double check the exact order of input arguments.

-> 인자의 순서 변경

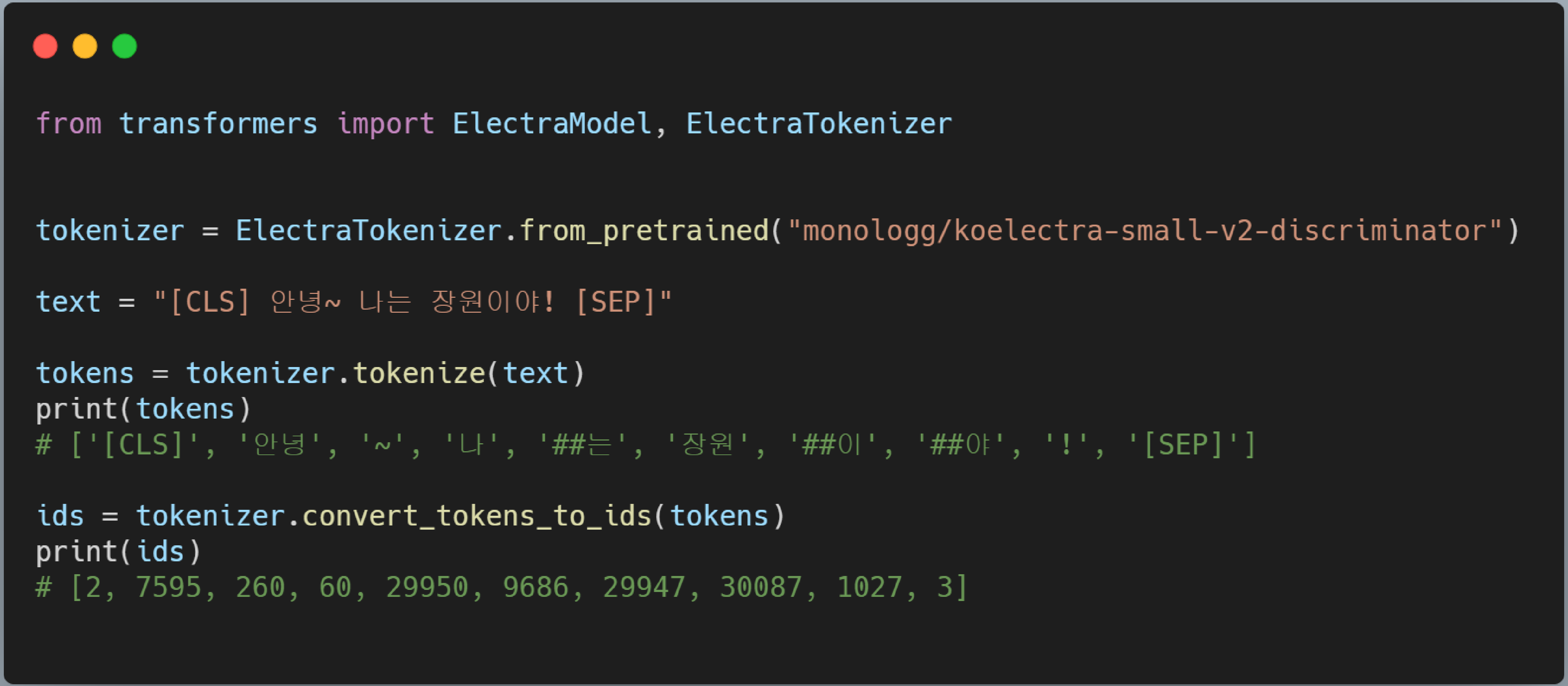
2. How to Use

Load Model & Tokenizer



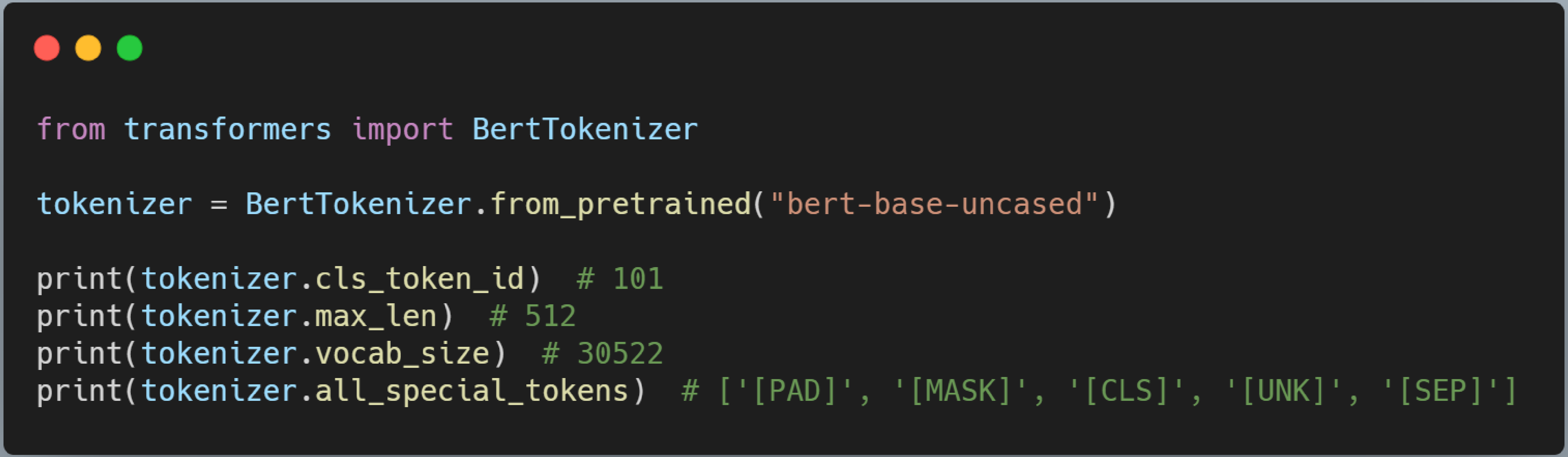
```python
from transformers import ElectraModel, ElectraTokenizer


model = ElectraModel.from_pretrained("monologg/koelectra-small-v2-discriminator")
tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-small-v2-discriminator")
```

Tokenizer API



```python
from transformers import ElectraModel, ElectraTokenizer


tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-small-v2-discriminator")

text = "[CLS] 안녕~ 나는 장원이야! [SEP]"

tokens = tokenizer.tokenize(text)
print(tokens)
# ['[CLS]', '안녕', '~', '나', '##는', '장원', '##이', '##야', '!', '[SEP]']

ids = tokenizer.convert_tokens_to_ids(tokens)
print(ids)
# [2, 7595, 260, 60, 29950, 9686, 29947, 30087, 1027, 3]
```

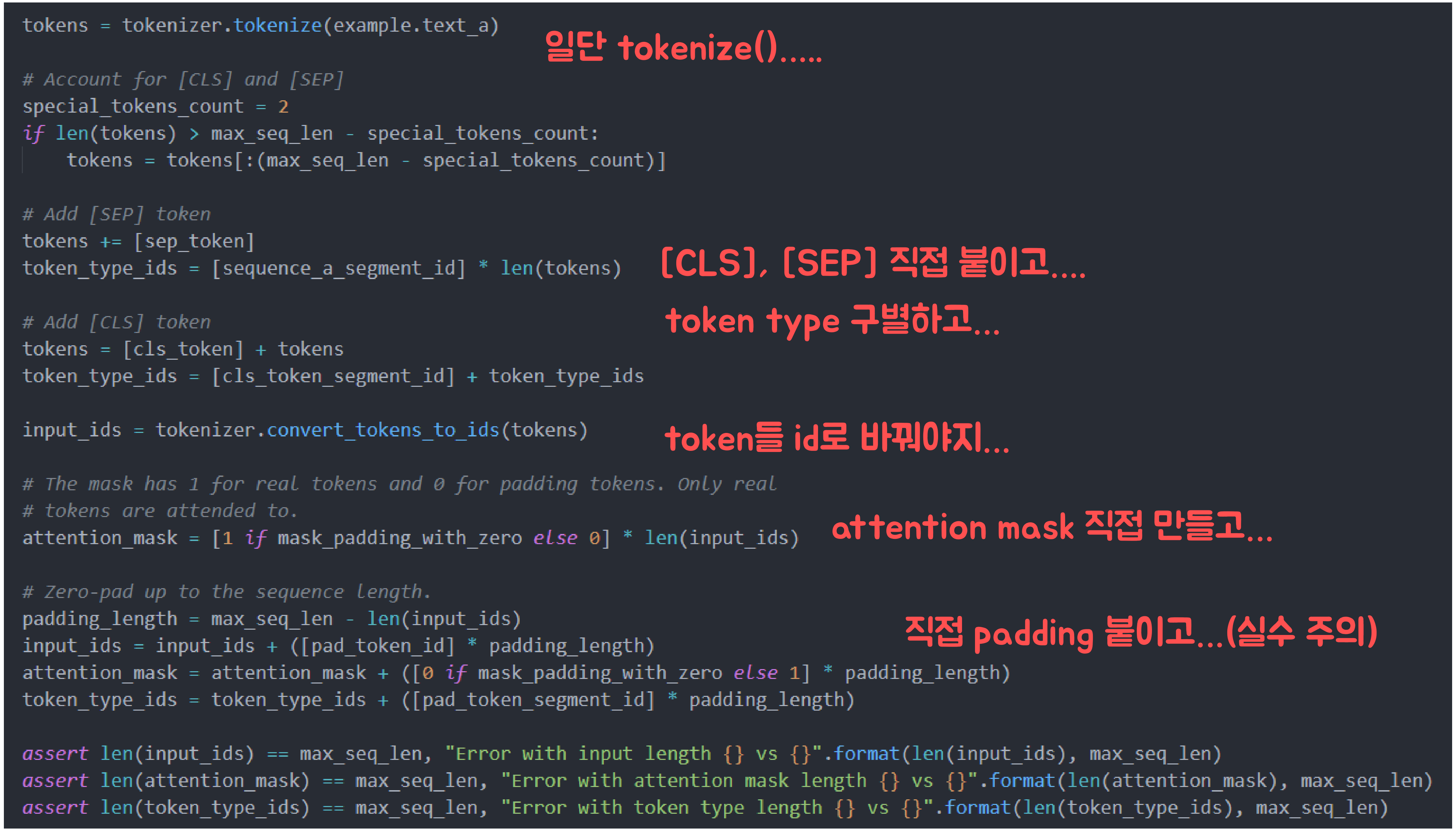
Tokenizer API



```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

print(tokenizer.cls_token_id)  # 101
print(tokenizer.max_len)  # 512
print(tokenizer.vocab_size)  # 30522
print(tokenizer.all_special_tokens)  # ['[PAD]', '[MASK]', '[CLS]', '[UNK]', '[SEP]']
```

-> 점점 다양해지고 있는 attribute!

하나하나 짜야한다니....구와악....



```python
tokens = tokenizer.tokenize(example.text_a)

# Account for [CLS] and [SEP]
special_tokens_count = 2
if len(tokens) > max_seq_len - special_tokens_count:
    tokens = tokens[:(max_seq_len - special_tokens_count)]

# Add [SEP] token
tokens += [sep_token]
token_type_ids = [sequence_a_segment_id] * len(tokens)

# Add [CLS] token
tokens = [cls_token] + tokens
token_type_ids = [cls_token_segment_id] + token_type_ids

input_ids = tokenizer.convert_tokens_to_ids(tokens)

# The mask has 1 for real tokens and 0 for padding tokens. Only real
# tokens are attended to.
attention_mask = [1 if mask_padding_with_zero else 0] * len(input_ids)

# Zero-pad up to the sequence length.
padding_length = max_seq_len - len(input_ids)
input_ids = input_ids + ([pad_token_id] * padding_length)
attention_mask = attention_mask + ([0 if mask_padding_with_zero else 1] * padding_length)
token_type_ids = token_type_ids + ([pad_token_segment_id] * padding_length)

assert len(input_ids) == max_seq_len, "Error with input length {} vs {}".format(len(input_ids), max_seq_len)
assert len(attention_mask) == max_seq_len, "Error with attention mask length {} vs {}".format(len(attention_mask), max_seq_len)
assert len(token_type_ids) == max_seq_len, "Error with token type length {} vs {}".format(len(token_type_ids), max_seq_len)
```

일단 tokenize().....

[CLS], [SEP] 직접 붙이고....

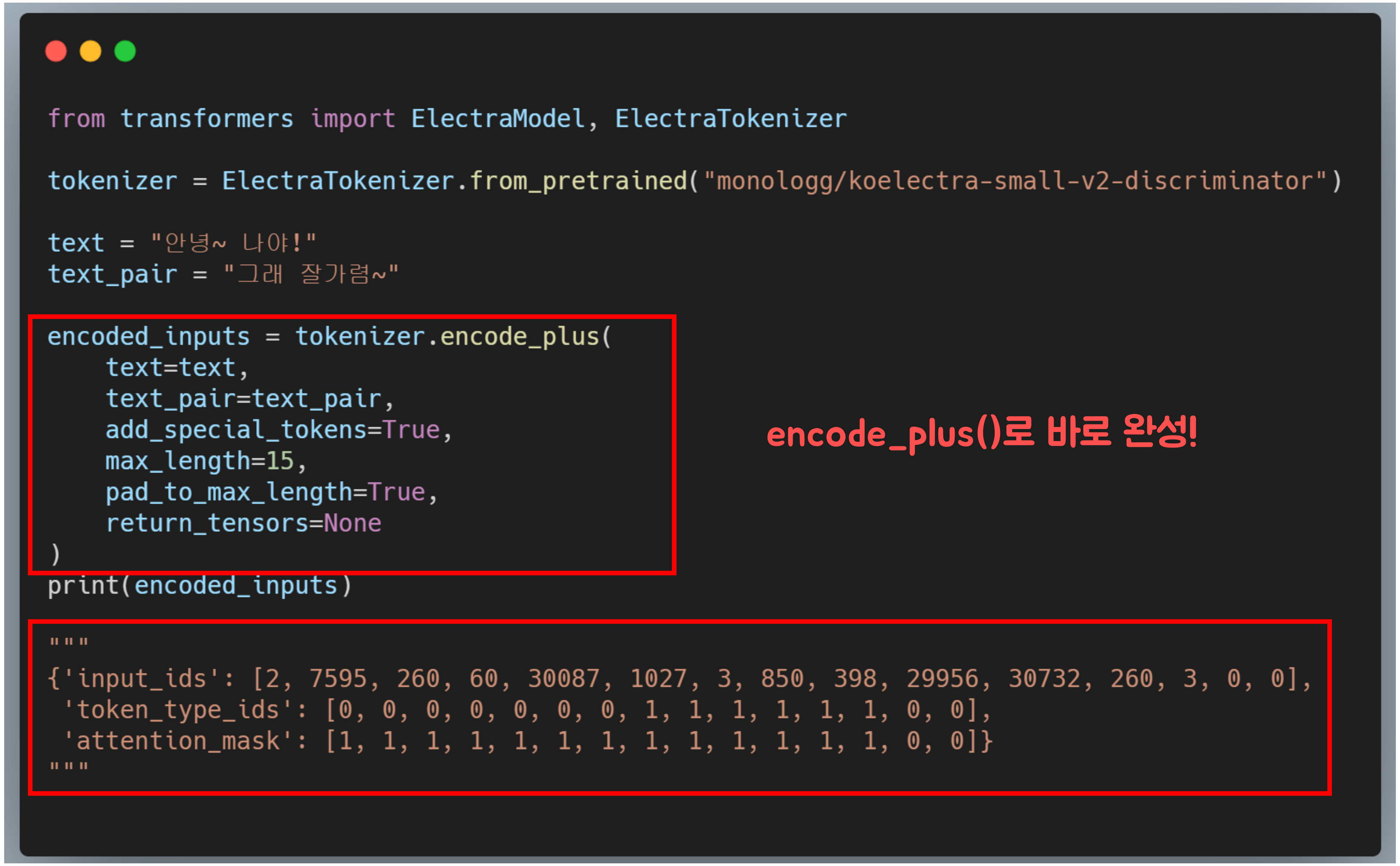
token type 구별하고...

token들 id로 바꿔야지...

attention mask 직접 만들고...

직접 padding 붙이고...(실수 주의)

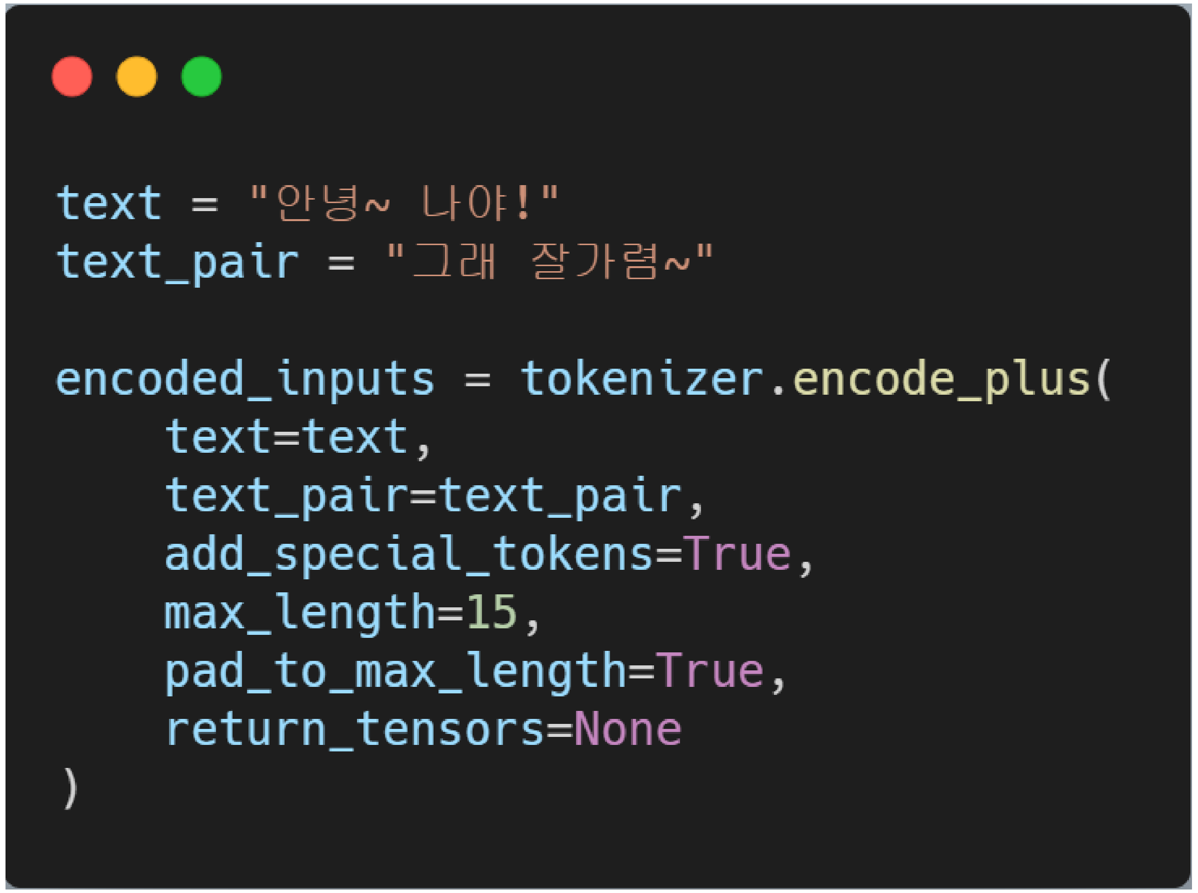
encode_plus()를 사용하면 매우 간편



```
from transformers import ElectraModel, ElectraTokenizer

tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-small-v2-discriminator")

text = "안녕~ 나야!"
text_pair = "그래 잘가렴~"

encoded_inputs = tokenizer.encode_plus(
    text=text,
    text_pair=text_pair,
    add_special_tokens=True,
    max_length=15,
    pad_to_max_length=True,
    return_tensors=None
)
print(encoded_inputs)

"""
{'input_ids': [2, 7595, 260, 60, 30087, 1027, 3, 850, 398, 29956, 30732, 260, 3, 0, 0],
 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]}
"""
```

encode_plus()로 바로 완성!

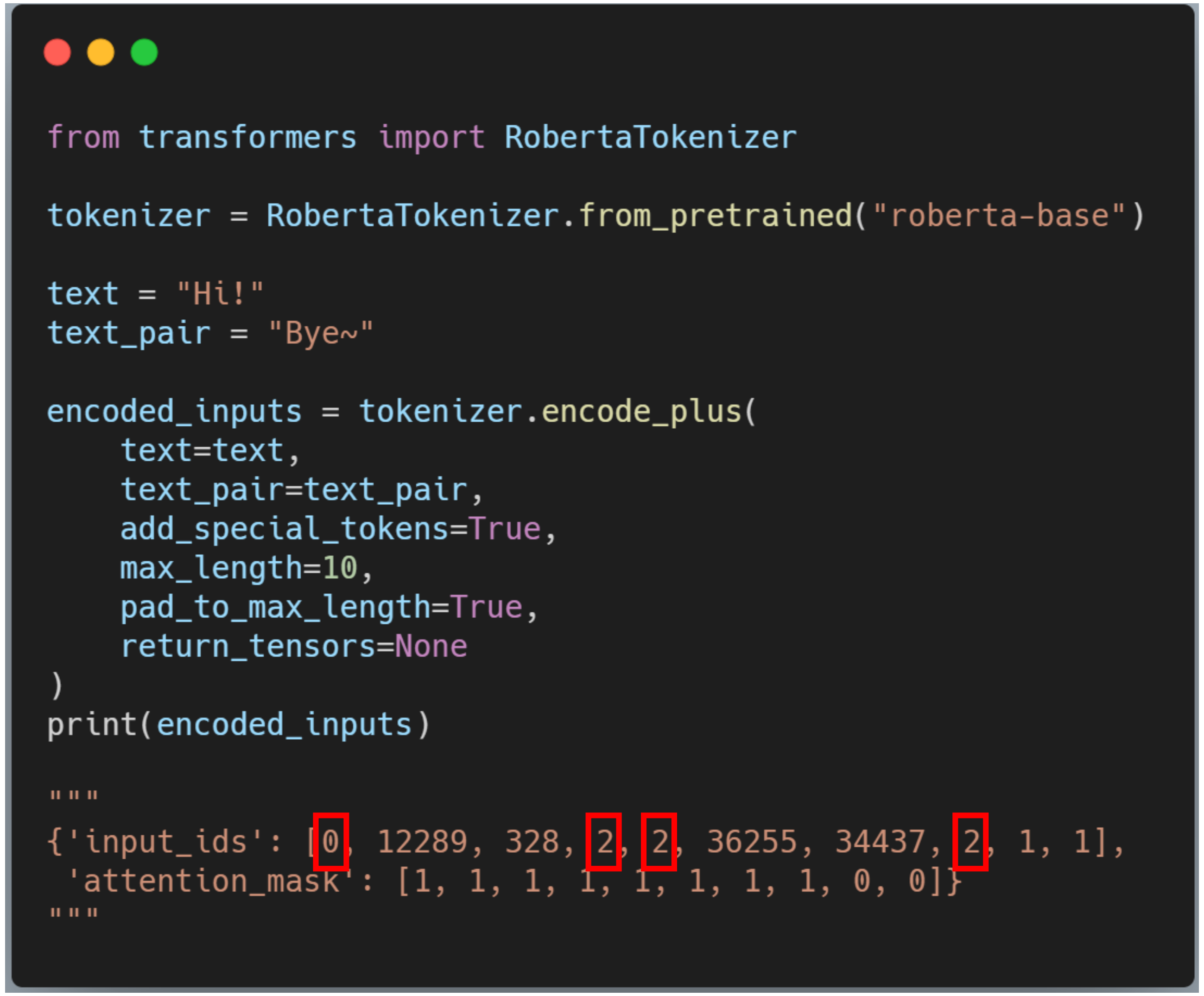
encode_plus()를 사용하면 매우 간편



```python
text = "안녕~ 나야!"
text_pair = "그래 잘가렴~"

encoded_inputs = tokenizer.encode_plus(
    text=text,
    text_pair=text_pair,
    add_special_tokens=True,
    max_length=15,
    pad_to_max_length=True,
    return_tensors=None
)
```

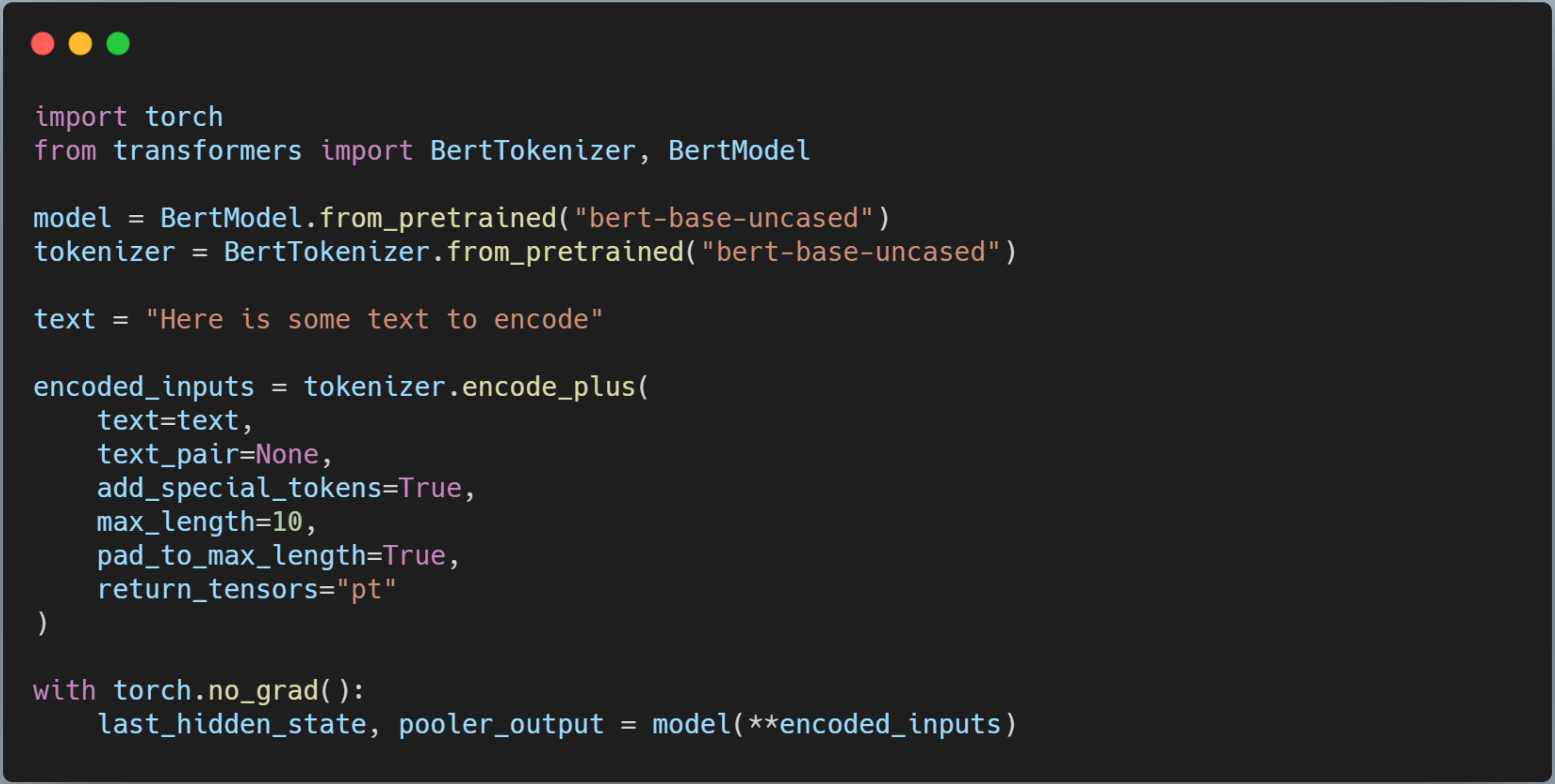
- text, text_pair
- add_special_tokens
  - [CLS] text_a [SEP]
  - [CLS] text_a [SEP] text_b [SEP]
- max_length, pad_to_max_length
- return_tensors
  - None (리스트), 'pt', 'tf'
  - 오늘부로 'np'도 지원됨 (#4585)

BERT와 RoBERTa의 차이 고민 안 해도 됨!



```python
from transformers import RobertaTokenizer

tokenizer = RobertaTokenizer.from_pretrained("roberta-base")

text = "Hi!"
text_pair = "Bye~"

encoded_inputs = tokenizer.encode_plus(
    text=text,
    text_pair=text_pair,
    add_special_tokens=True,
    max_length=10,
    pad_to_max_length=True,
    return_tensors=None
)
print(encoded_inputs)

"""
{'input_ids': [0, 12289, 328, 2, 2, 36255, 34437, 2, 1, 1],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 0, 0]}
"""
```

- special token 자동 처리
  - <s> text_a </s> </s> text_b </s>
- pad_token_id = 1
- token_type_ids를 쓰지 않는 것도 자동 처리



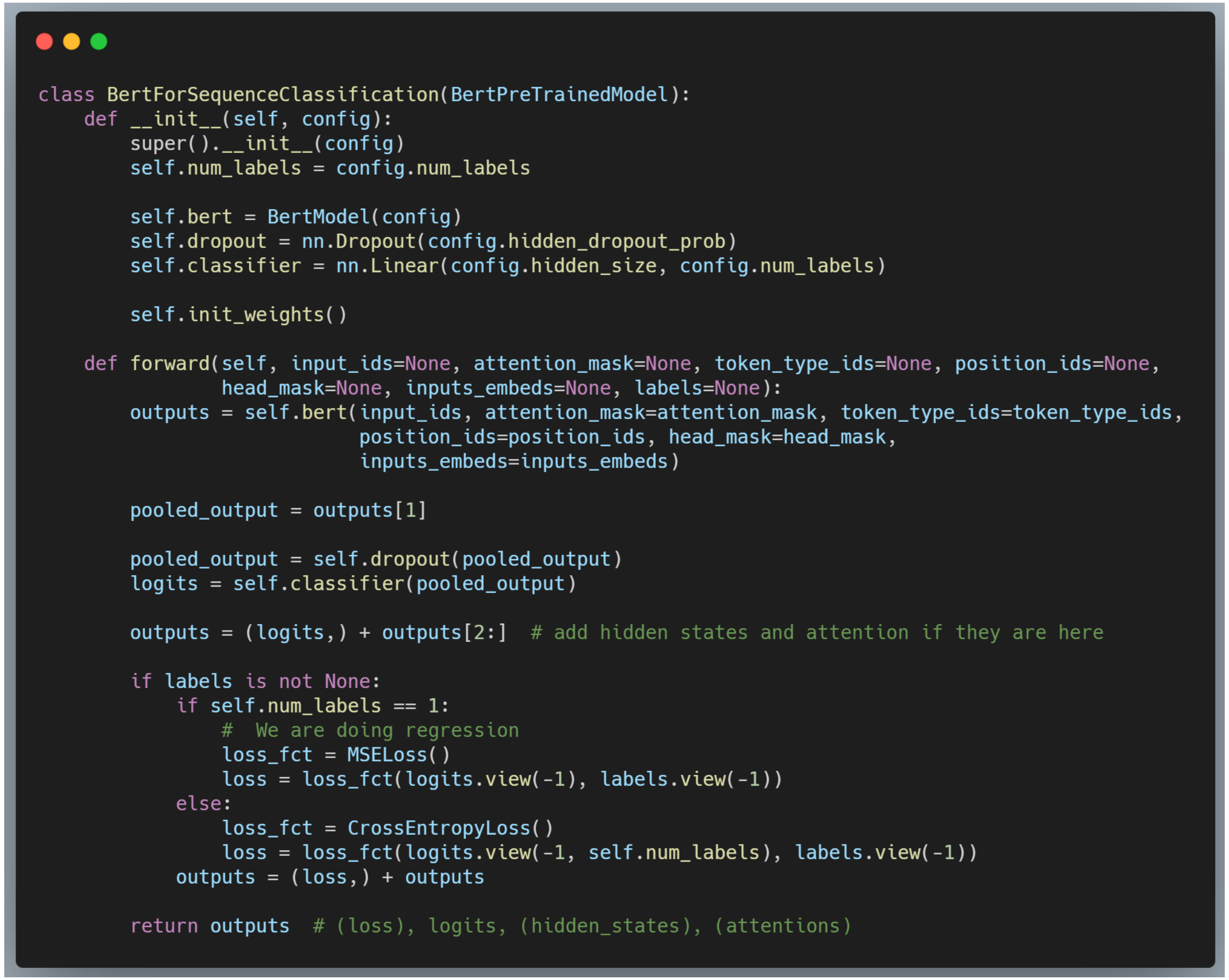
```python
import torch
from transformers import BertTokenizer, BertModel

model = BertModel.from_pretrained("bert-base-uncased")
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

text = "Here is some text to encode"

encoded_inputs = tokenizer.encode_plus(
    text=text,
    text_pair=None,
    add_special_tokens=True,
    max_length=10,
    pad_to_max_length=True,
    return_tensors="pt"
)

with torch.no_grad():
    last_hidden_state, pooler_output = model(**encoded_inputs)
```

Finetuning Model

BertForSequenceClassification

BertForTokenClassification

BertForQuestionAnswering



```python
class BertForSequenceClassification(BertPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels

        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)

        self.init_weights()

    def forward(self, input_ids=None, attention_mask=None, token_type_ids=None, position_ids=None,
                head_mask=None, inputs_embeds=None, labels=None):
        outputs = self.bert(input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids,
                            position_ids=position_ids, head_mask=head_mask,
                            inputs_embeds=inputs_embeds)

        pooled_output = outputs[1]

        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)

        outputs = (logits,) + outputs[2:]  # add hidden states and attention if they are here

        if labels is not None:
            if self.num_labels == 1:
                #  We are doing regression
                loss_fct = MSELoss()
                loss = loss_fct(logits.view(-1), labels.view(-1))
            else:
                loss_fct = CrossEntropyLoss()
                loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
            outputs = (loss,) + outputs

        return outputs  # (loss), logits, (hidden_states), (attentions)
```

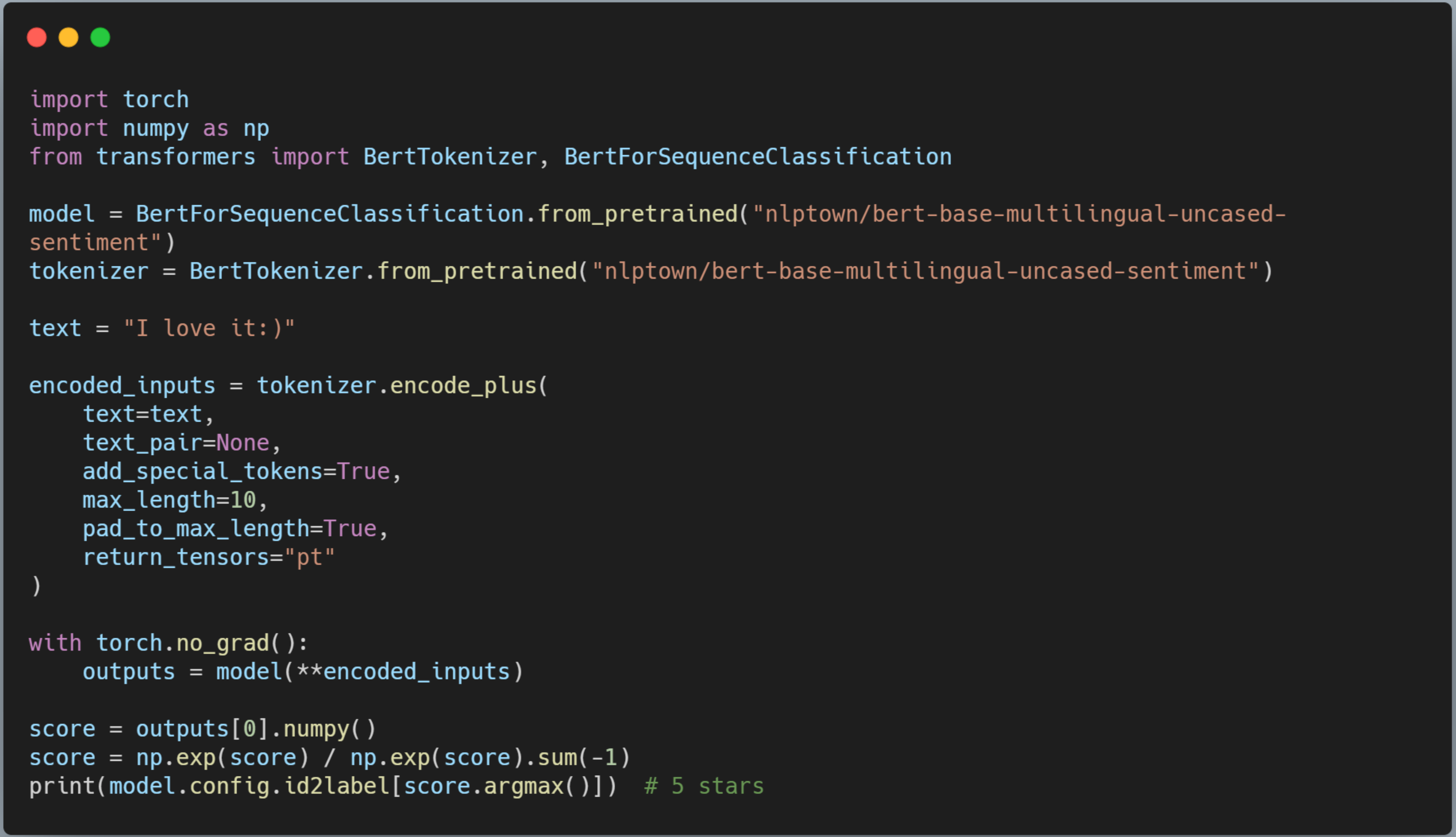
Finetuning Model

BertForSequenceClassification

BertForTokenClassification

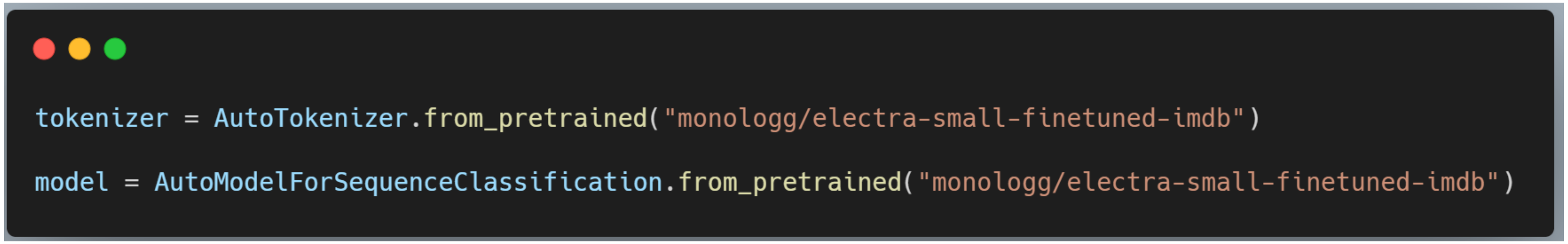
BertForQuestionAnswering

-> Prototyping, Example Code 만들 때 유용

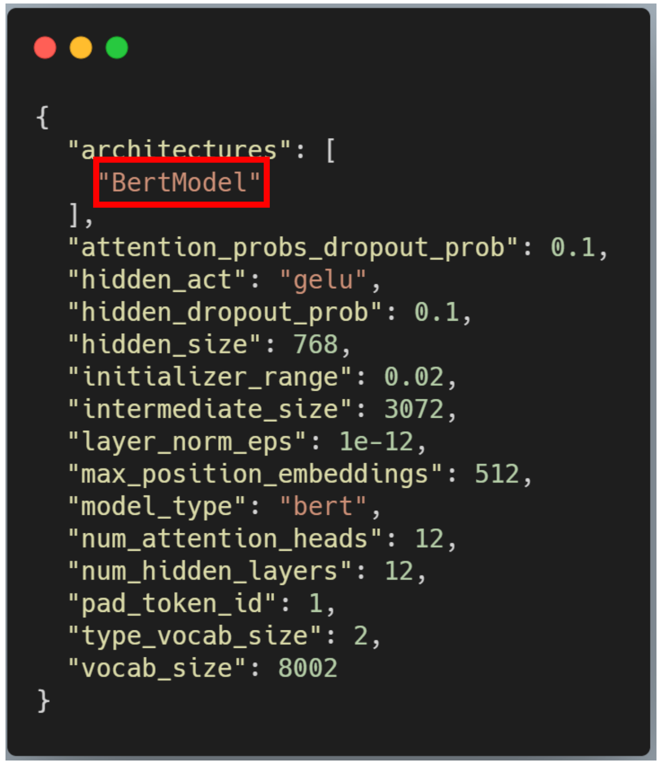


```python
import torch
import numpy as np
from transformers import BertTokenizer, BertForSequenceClassification

model = BertForSequenceClassification.from_pretrained("nlptown/bert-base-multilingual-uncased-
sentiment")
tokenizer = BertTokenizer.from_pretrained("nlptown/bert-base-multilingual-uncased-sentiment")

text = "I love it:)"

encoded_inputs = tokenizer.encode_plus(
    text=text,
    text_pair=None,
    add_special_tokens=True,
    max_length=10,
    pad_to_max_length=True,
    return_tensors="pt"
)

with torch.no_grad():
    outputs = model(**encoded_inputs)

score = outputs[0].numpy()
score = np.exp(score) / np.exp(score).sum(-1)
print(model.config.id2label[score.argmax()])  # 5 stars
```

AutoModel, AutoTokenizer



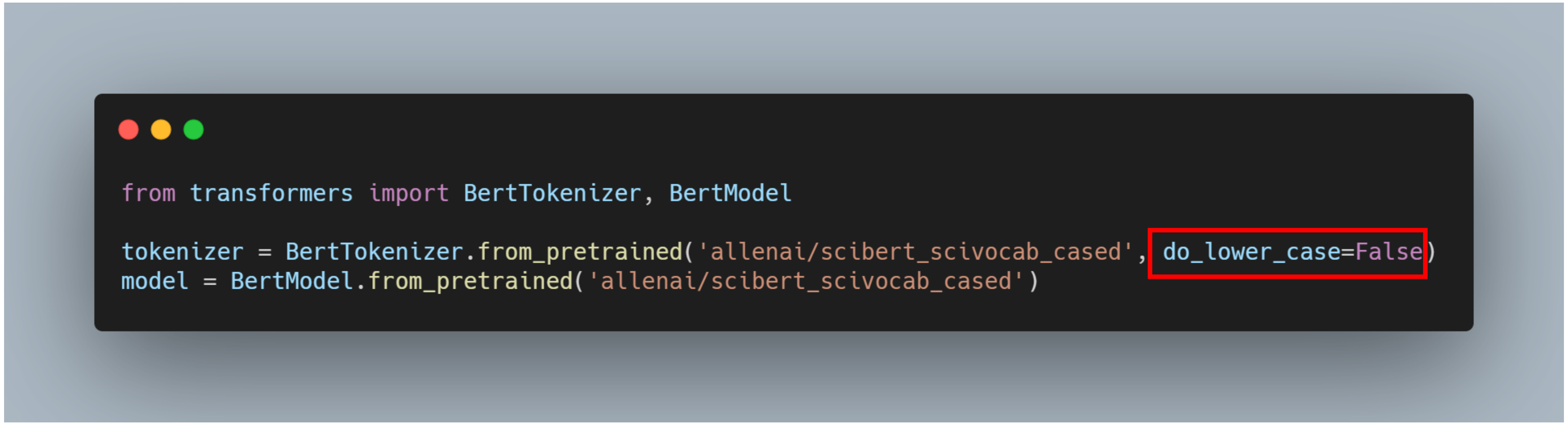
```
tokenizer = AutoTokenizer.from_pretrained("monologg/electra-small-finetuned-imdb")

model = AutoModelForSequenceClassification.from_pretrained("monologg/electra-small-finetuned-imdb")
```



```
{
  "architectures": [
    "BertModel"
  ],
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 1,
  "type_vocab_size": 2,
  "vocab_size": 8002
}
```

- config.json을 보고 어떤 모델인지 예측
- 옛 버전에서는 예기치 못한 이슈들이 있었음
- 개인적으로는 비추 (역시 명확하게 명시해주는 것이 좋지 않나…)

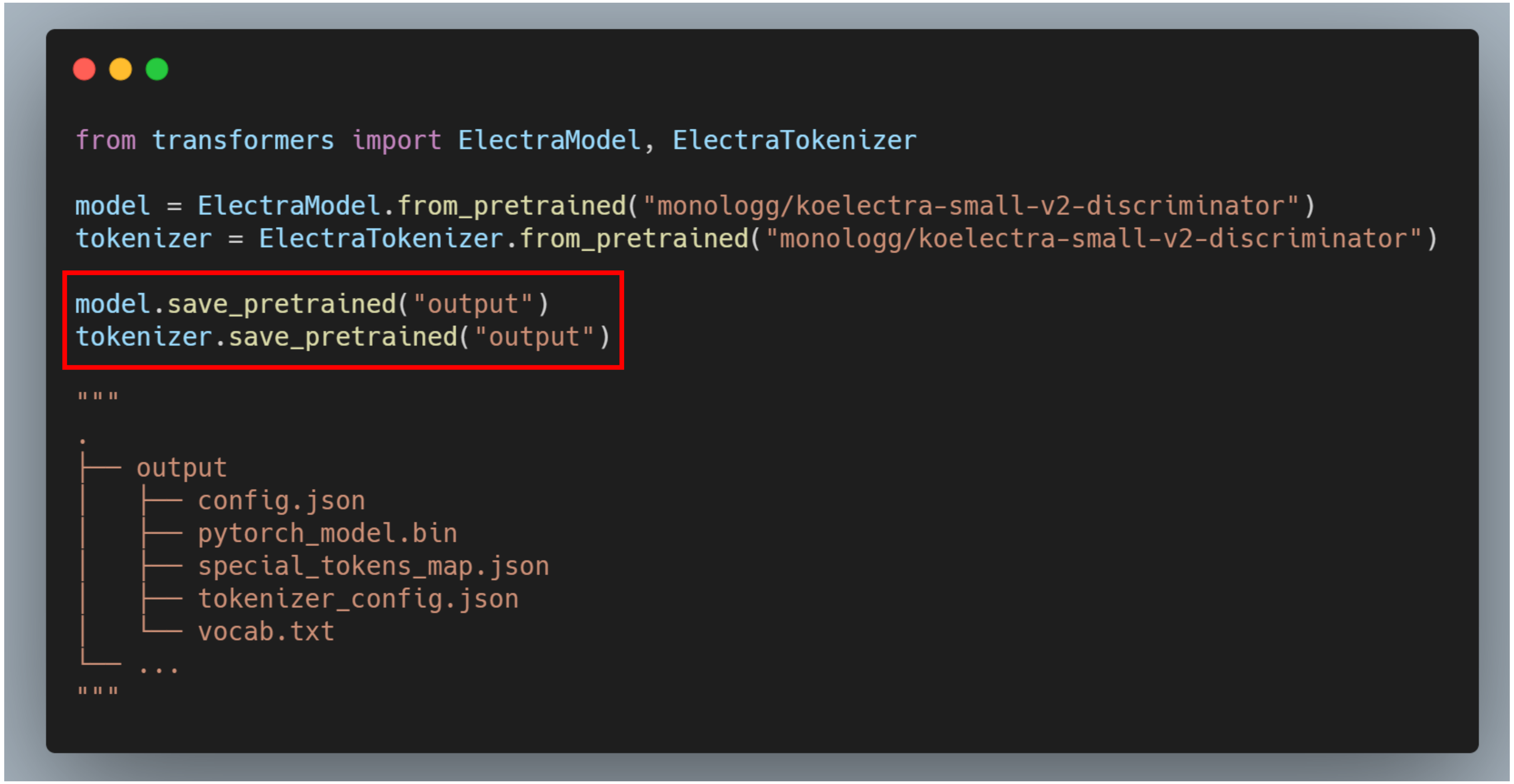
Tokenizer 사용시 do_lower_case 주의하라!!



```
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained('allenai/scibert_scivocab_cased', do_lower_case=False)
model = BertModel.from_pretrained('allenai/scibert_scivocab_cased')
```

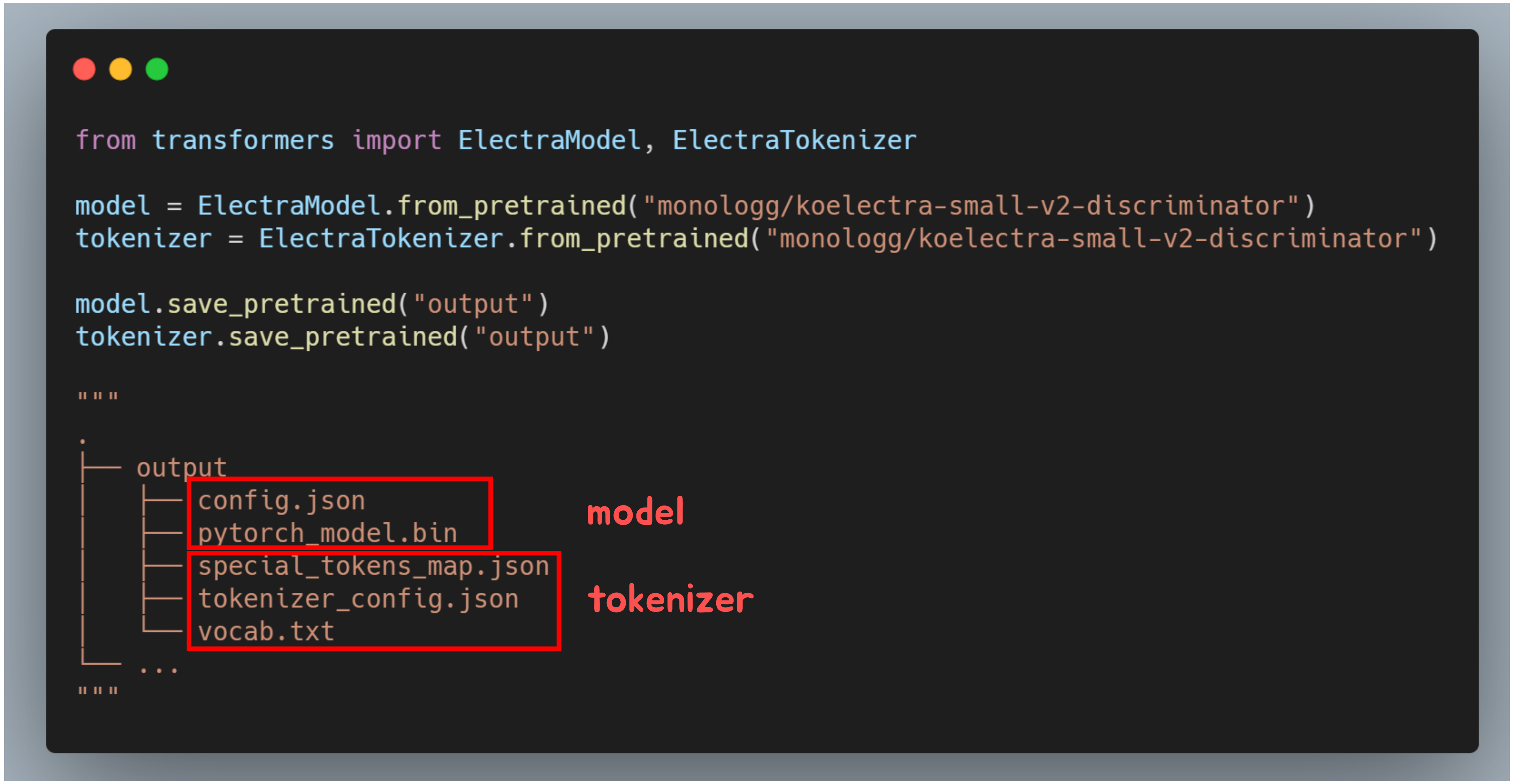
- 한국어의 경우 대부분 cased 모델
- 여러 코드들에서도 실수가 많이 나오는 부분 (코드 가져다 쓸 때도 주의하면 좋음!)
- tokenizer_config.json에 명시하면 해결됨!

Save Model & Tokenizer



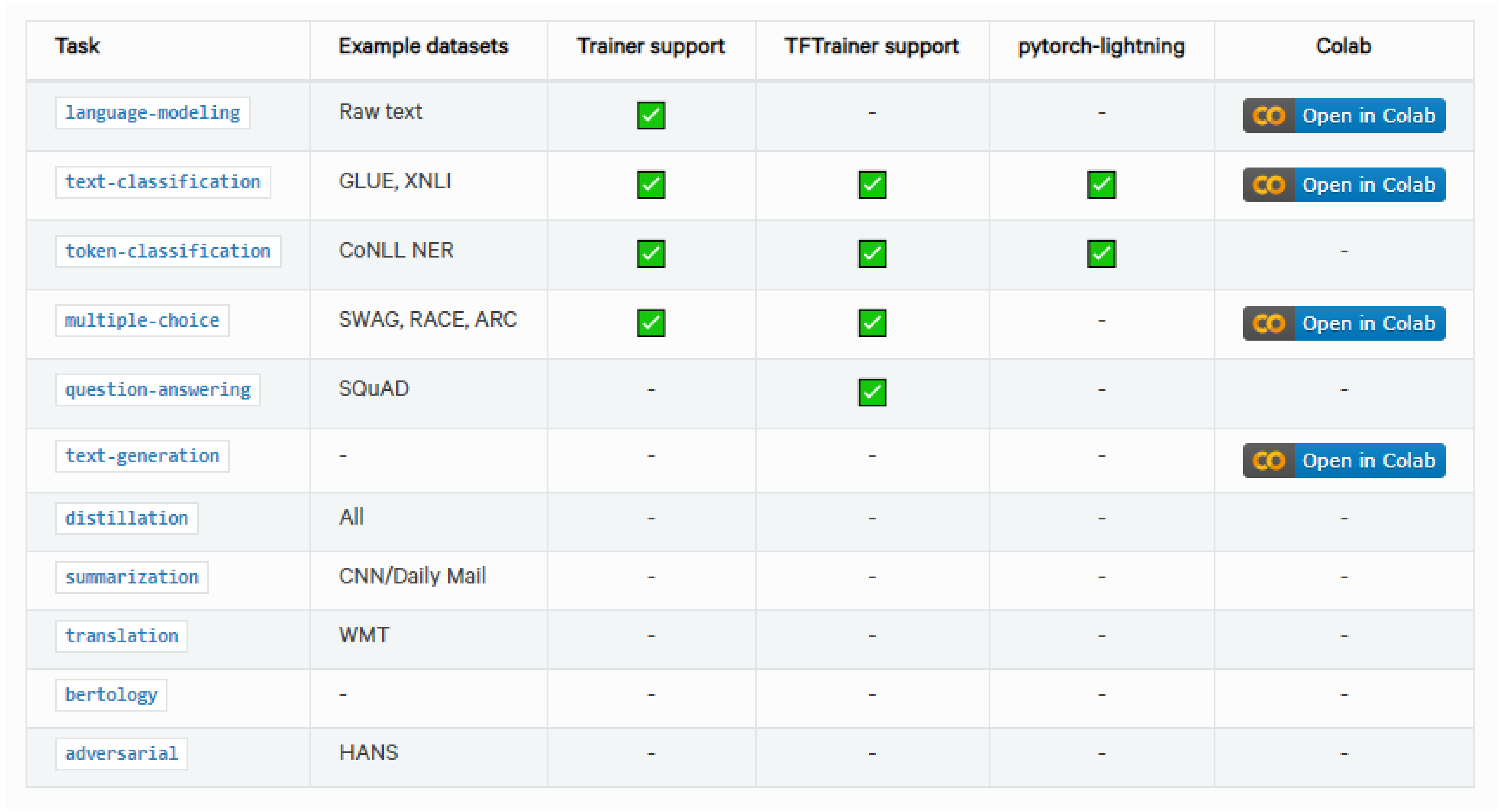
```
from transformers import ElectraModel, ElectraTokenizer

model = ElectraModel.from_pretrained("monologg/koelectra-small-v2-discriminator")
tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-small-v2-discriminator")

model.save_pretrained("output")
tokenizer.save_pretrained("output")

"""
.
├── output
│   ├── config.json
│   ├── pytorch_model.bin
│   ├── special_tokens_map.json
│   ├── tokenizer_config.json
│   └── vocab.txt
└── ...
"""
```

Save Model & Tokenizer



```python
from transformers import ElectraModel, ElectraTokenizer

model = ElectraModel.from_pretrained("monologg/koelectra-small-v2-discriminator")
tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-small-v2-discriminator")

model.save_pretrained("output")
tokenizer.save_pretrained("output")

"""
.
├── output
│   ├── config.json
│   ├── pytorch_model.bin
│   ├── special_tokens_map.json
│   ├── tokenizer_config.json
│   └── vocab.txt
└── ...
"""
```

3. Examples

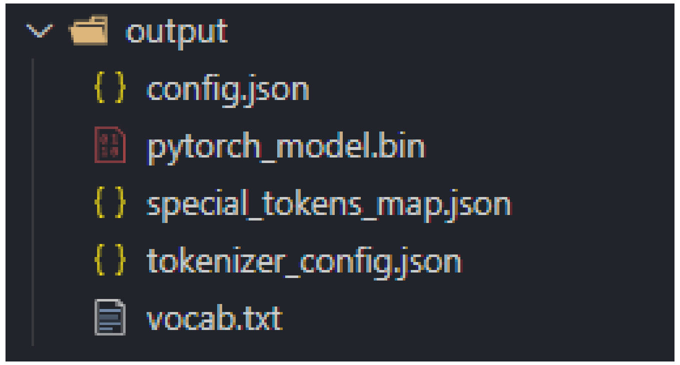


Task	Example datasets	Trainer support	TFTrainer support	pytorch-lightning	Colab
`language-modeling`	Raw text	✓	-	-	Open in Colab
`text-classification`	GLUE, XNLI	✓	✓	✓	Open in Colab
`token-classification`	CoNLL NER	✓	✓	✓	-
`multiple-choice`	SWAG, RACE, ARC	✓	✓	-	Open in Colab
`question-answering`	SQuAD	-	✓	-	-
`text-generation`	-	-	-	-	Open in Colab
`distillation`	All	-	-	-	-
`summarization`	CNN/Daily Mail	-	-	-	-
`translation`	WMT	-	-	-	-
`bertology`	-	-	-	-	-
`adversarial`	HANS	-	-	-	-

https://github.com/huggingface/transformers/blob/master/examples/README.md

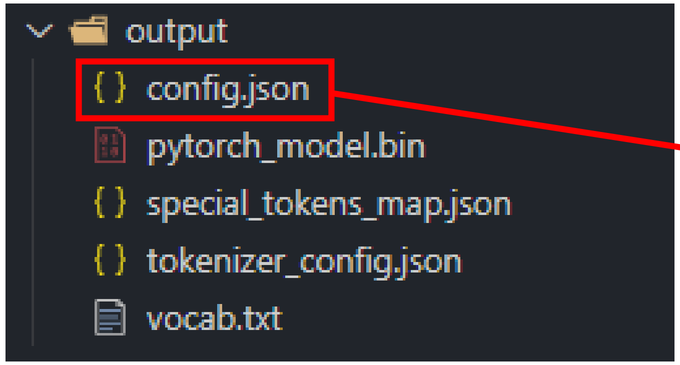
4. Convert TF ckpt

Transformers Directory

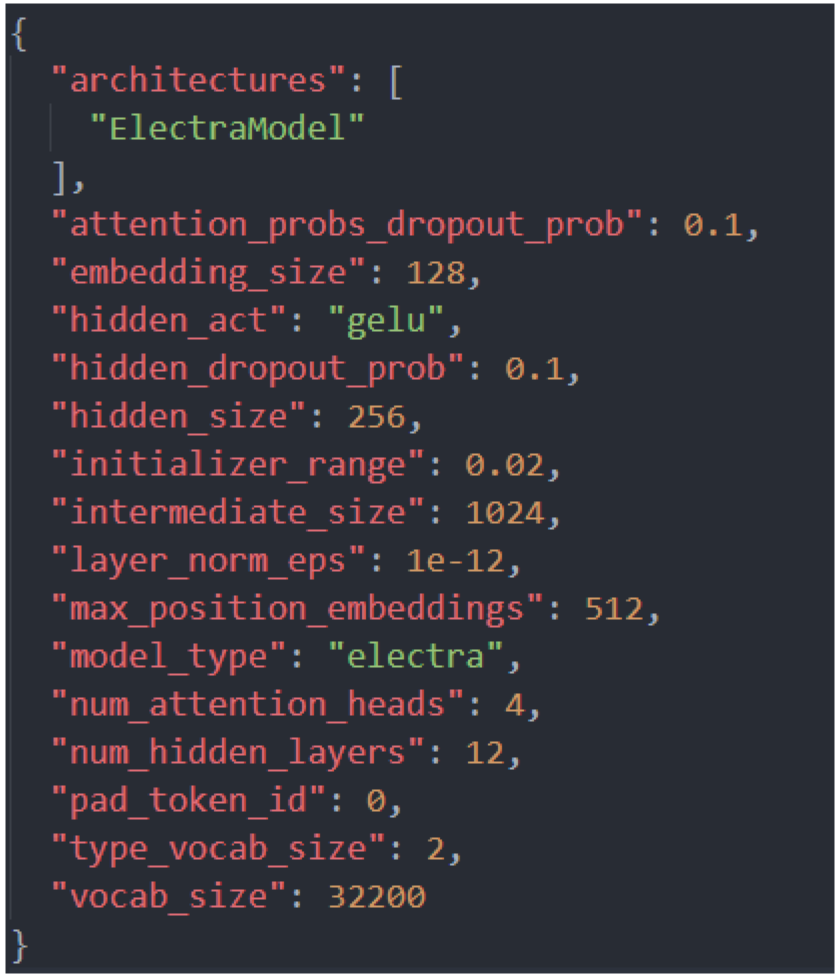


```
output
    config.json
    pytorch_model.bin
    special_tokens_map.json
    tokenizer_config.json
    vocab.txt
```

Transformers Directory

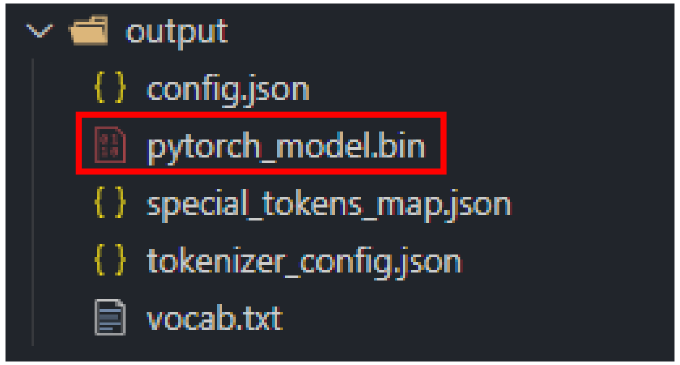


1. config.json



```
{
  "architectures": [
    "ElectraModel"
  ],
  "attention_probs_dropout_prob": 0.1,
  "embedding_size": 128,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 256,
  "initializer_range": 0.02,
  "intermediate_size": 1024,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "electra",
  "num_attention_heads": 4,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "type_vocab_size": 2,
  "vocab_size": 32200
}
```

Transformers Directory

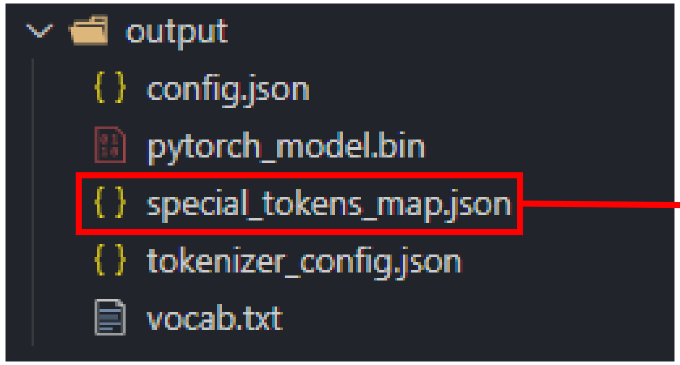


output
config.json
pytorch_model.bin
special_tokens_map.json
tokenizer_config.json
vocab.txt

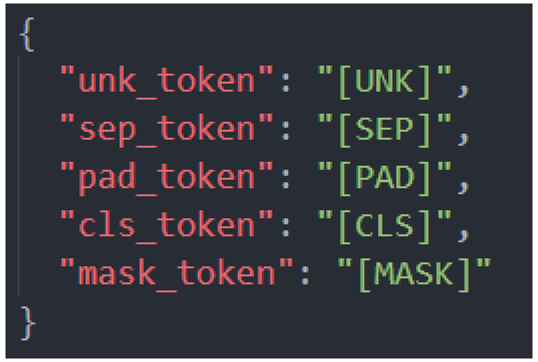
2. pytorch_model.bin

== pretrained weight

Transformers Directory

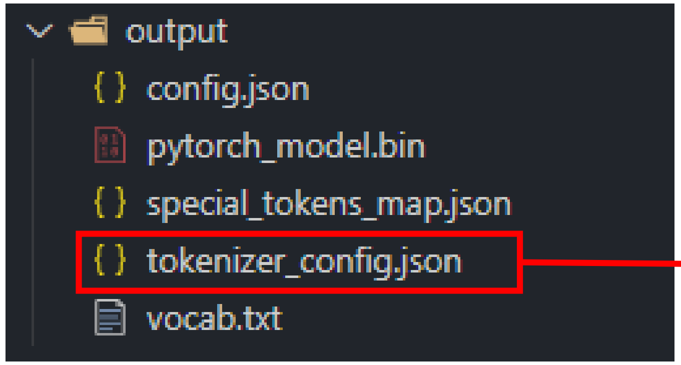

```
output
  config.json
  pytorch_model.bin
  special_tokens_map.json
  tokenizer_config.json
  vocab.txt
```

3. special_tokens_map.json


```json
{
  "unk_token": "[UNK]",
  "sep_token": "[SEP]",
  "pad_token": "[PAD]",
  "cls_token": "[CLS]",
  "mask_token": "[MASK]"
}
```

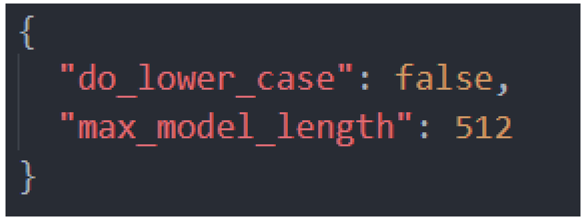
내가 추가적인 special token을 정의하지 않는 이상 필요 X

Transformers Directory



```
output
  config.json
  pytorch_model.bin
  special_tokens_map.json
  tokenizer_config.json
  vocab.txt
```

4. tokenizer_config.json

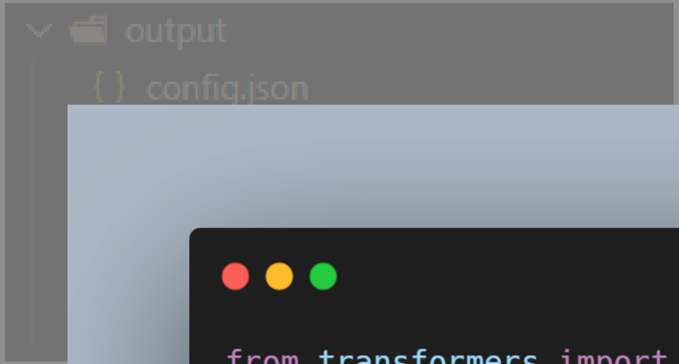


```
{
  "do_lower_case": false,
  "max_model_length": 512
}
```

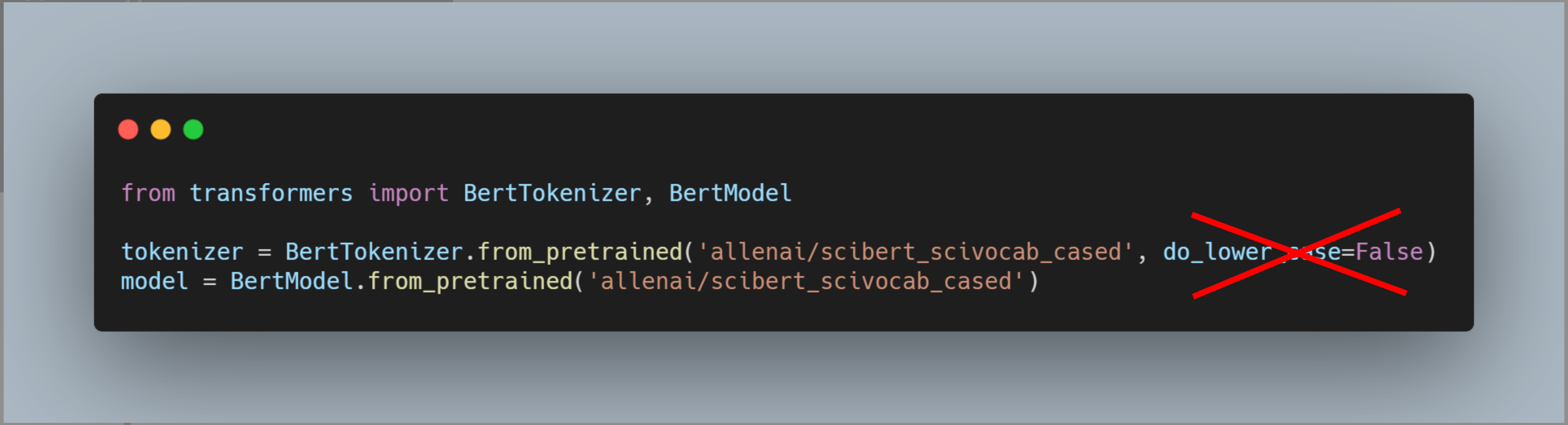
여기서 do_lower_case=False로 정의하면

tokenizer.from_pretrained()에서 따로 호출할 필요 없음!

Transformers Directory



4. tokenizer_config.json

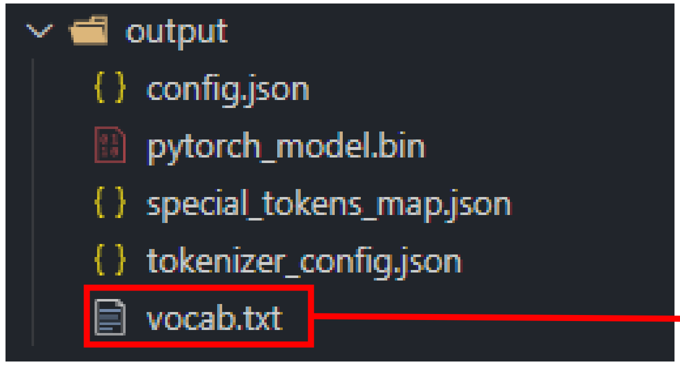


```
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained('allenai/scibert_scivocab_cased', ~~do_lower_case=False~~)
model = BertModel.from_pretrained('allenai/scibert_scivocab_cased')
```

여기서 do_lower_case=False로 정의하면

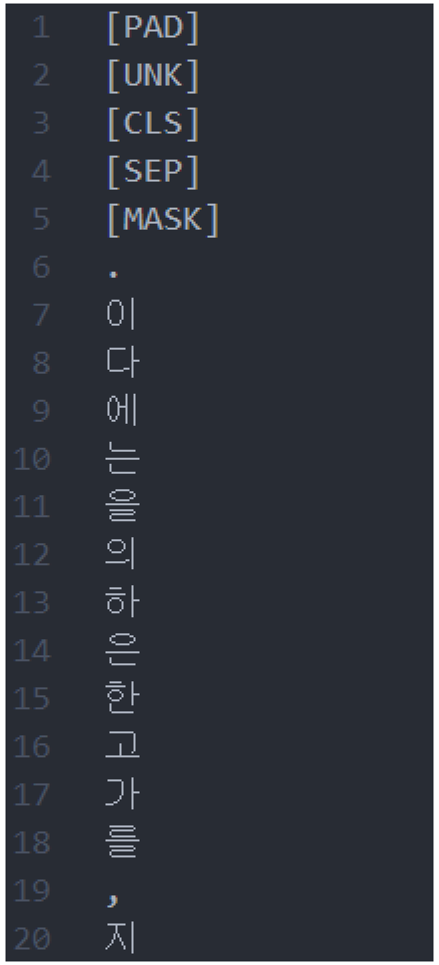
tokenizer.from_pretrained()에서 따로 호출할 필요 없음!

Transformers Directory



```
output
  config.json
  pytorch_model.bin
  special_tokens_map.json
  tokenizer_config.json
  vocab.txt
```

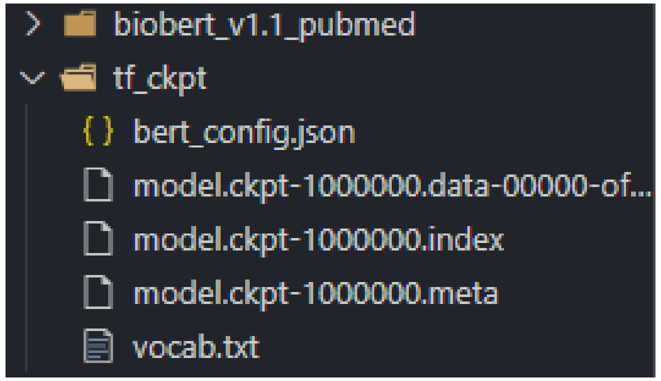
5. vocab.txt



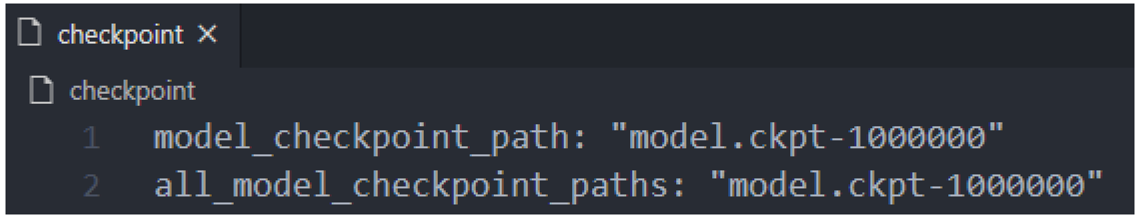
```
[PAD]
[UNK]
[CLS]
[SEP]
[MASK]
.
이
다
에
는
을
의
하
은
한
고
가
를
,
지
```

BioBERT를 직접 변환해보자

1. checkpoint 없을 시 직접 만들기



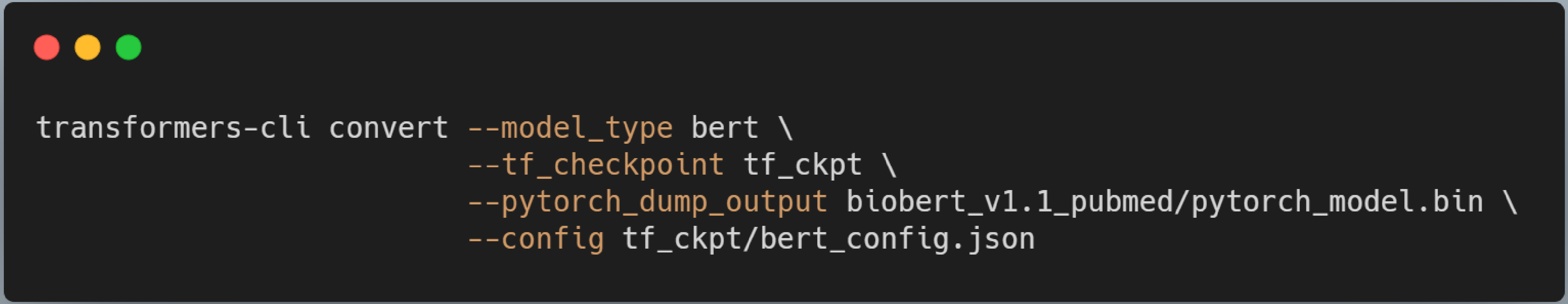
```
> biobert_v1.1_pubmed
v tf_ckpt
    bert_config.json
    model.ckpt-1000000.data-00000-of...
    model.ckpt-1000000.index
    model.ckpt-1000000.meta
    vocab.txt
```



```
model_checkpoint_path: "model.ckpt-1000000"
all_model_checkpoint_paths: "model.ckpt-1000000"
```

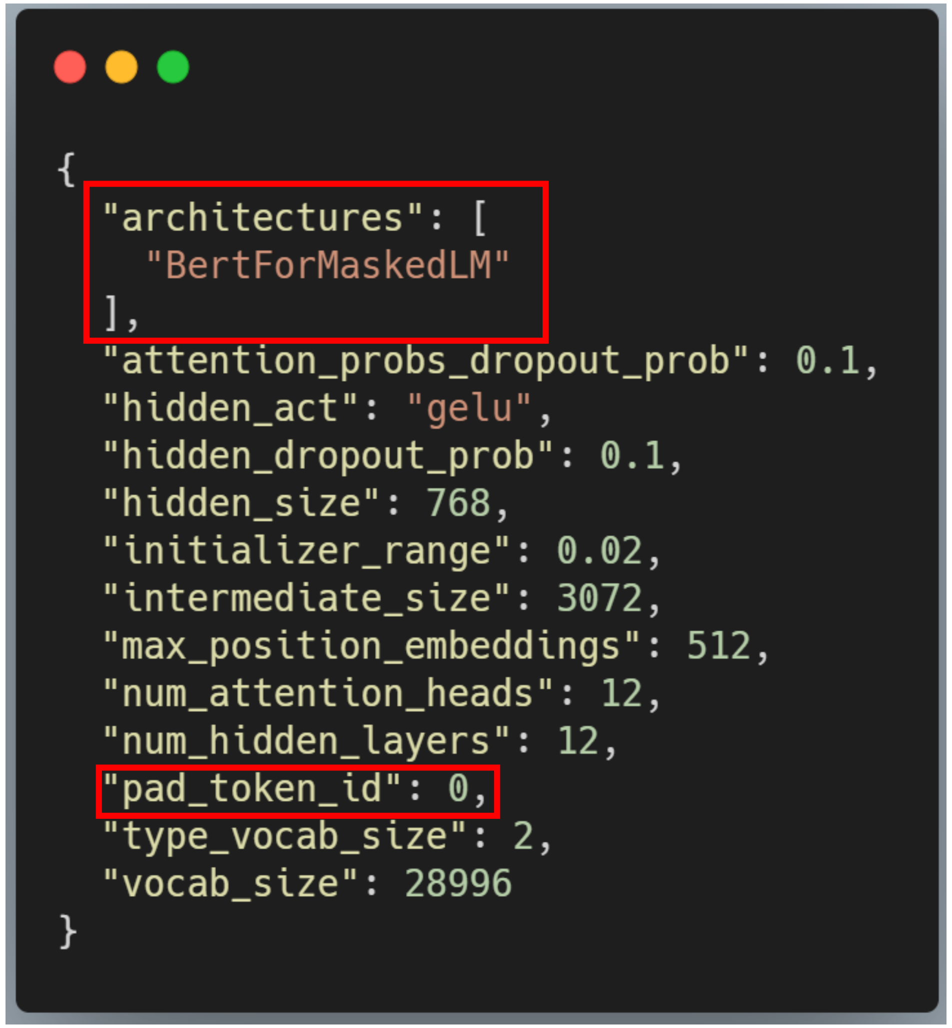
-> checkpoint 파일이 없네.... 직접 만들어야겠다ㅠ

2. convert command

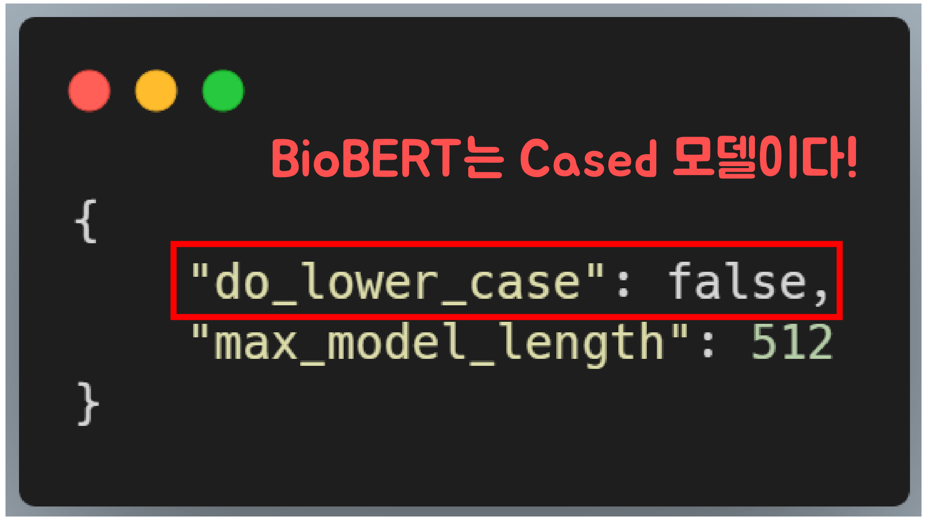


```
transformers-cli convert --model_type bert \
                         --tf_checkpoint tf_ckpt \
                         --pytorch_dump_output biobert_v1.1_pubmed/pytorch_model.bin \
                         --config tf_ckpt/bert_config.json
```

3. config.json, tokenizer_config.json 만들기



```
{
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "type_vocab_size": 2,
  "vocab_size": 28996
}
```



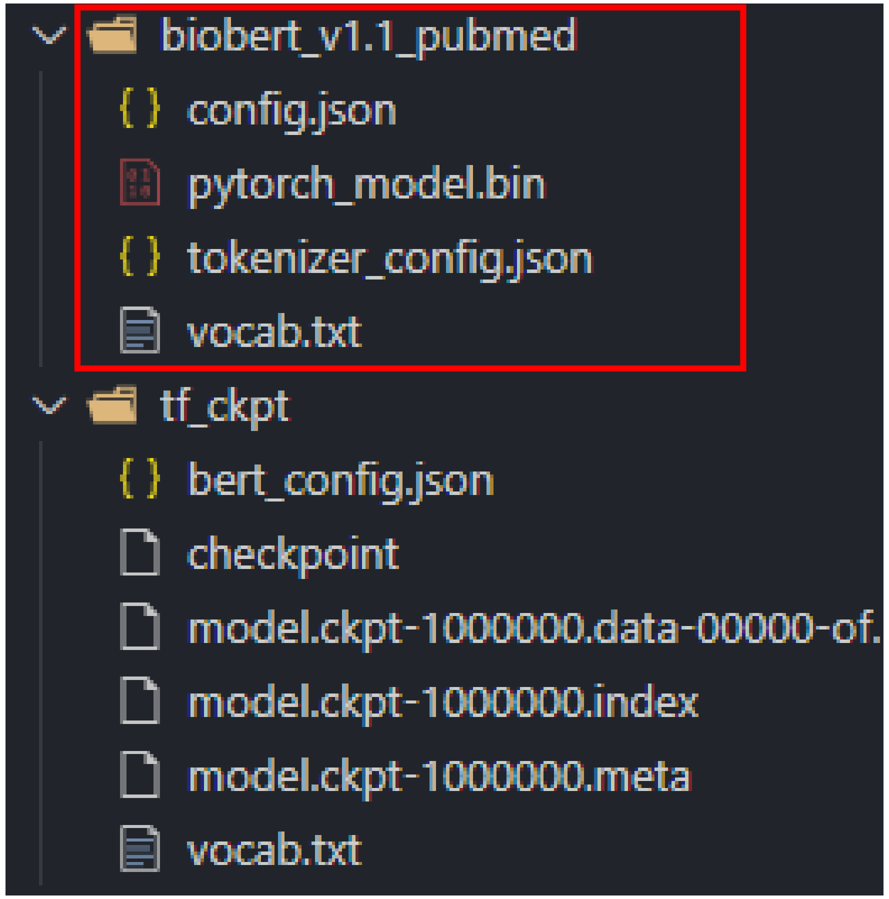
BioBERT는 Cased 모델이다!

```
{
    "do_lower_case": false,
    "max_model_length": 512
}
```

config.json, tokenizer_config.json 특히 주의하라

- API가 생각보다 자주 바뀐다 ~~(max_model_length로 대체 언제 바뀌었...)~~
- https://github.com/huggingface/transformers/releases
- https://huggingface.co/models
  - bert-base-uncased 등을 검색해서 복붙하고 시작하는게 베스트

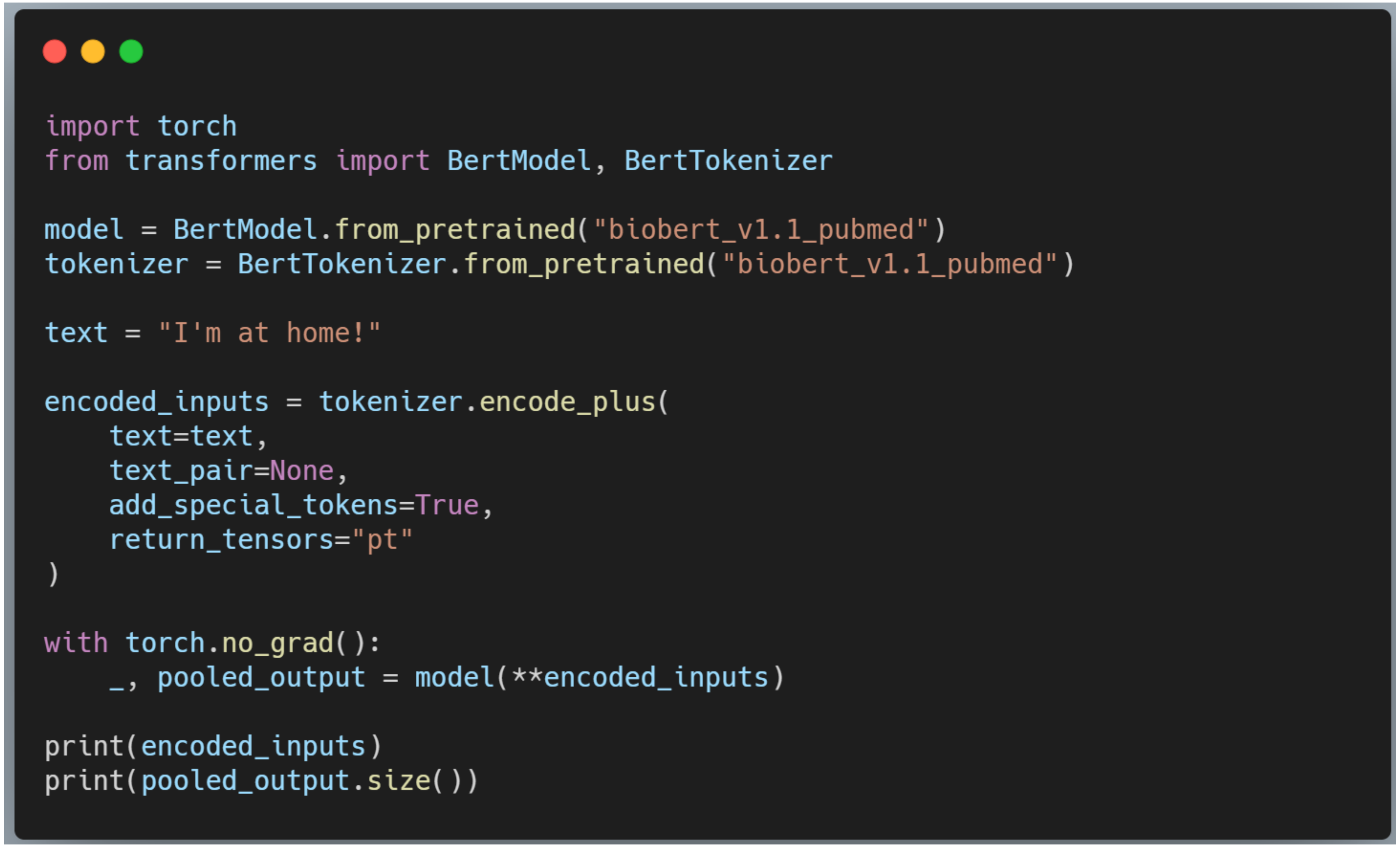
4. vocab.txt 옮기기



biobert_v1.1_pubmed
- config.json
- pytorch_model.bin
- tokenizer_config.json
- vocab.txt

tf_ckpt
- bert_config.json
- checkpoint
- model.ckpt-1000000.data-00000-of.
- model.ckpt-1000000.index
- model.ckpt-1000000.meta
- vocab.txt

5. Testing



```
import torch
from transformers import BertModel, BertTokenizer

model = BertModel.from_pretrained("biobert_v1.1_pubmed")
tokenizer = BertTokenizer.from_pretrained("biobert_v1.1_pubmed")

text = "I'm at home!"

encoded_inputs = tokenizer.encode_plus(
    text=text,
    text_pair=None,
    add_special_tokens=True,
    return_tensors="pt"
)

with torch.no_grad():
    _, pooled_output = model(**encoded_inputs)

print(encoded_inputs)
print(pooled_output.size())
```

https://monologg.kr/2020/05/01/transformers-porting/

5. Tokenizer 직접 만들기

https://github.com/huggingface/transformers/blob/master/templates/adding_a_new_model/README.md

https://github.com/huggingface/transformers/blob/master/templates/adding_a_new_model/tokenization_xxx.py

좋은 Reference

1. Wordpiece -> Bert

https://github.com/huggingface/transformers/blob/master/src/transformers/tokenization_bert.py

2. Sentencepiece -> XLNet

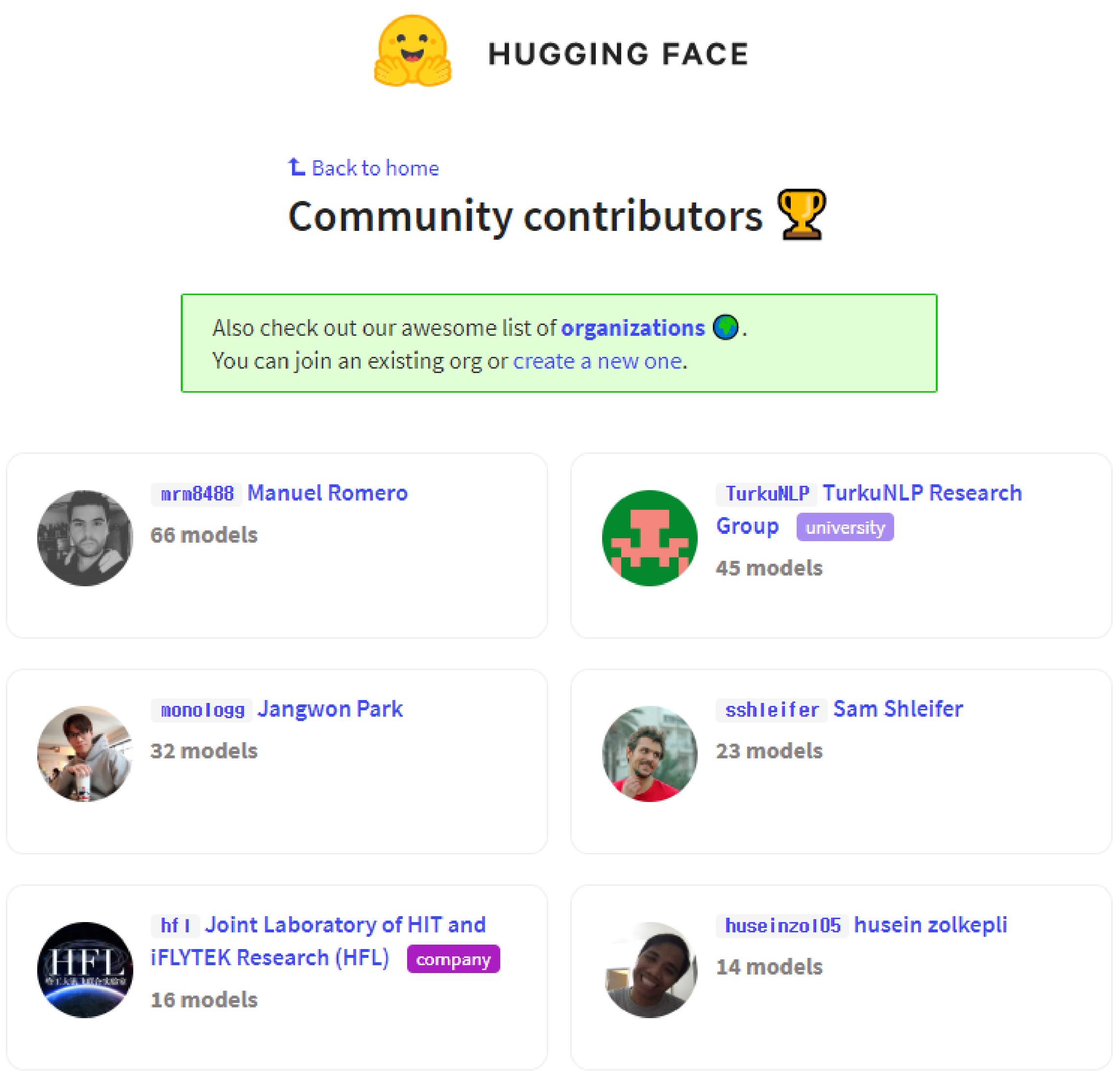
https://github.com/huggingface/transformers/blob/master/src/transformers/tokenization_xlnet.py

3. Mecab -> BertJapanese

https://github.com/huggingface/transformers/blob/master/src/transformers/tokenization_bert_japanese.py

6. Model S3 Upload

용량 무제한.... 공짜 너무 좋아....



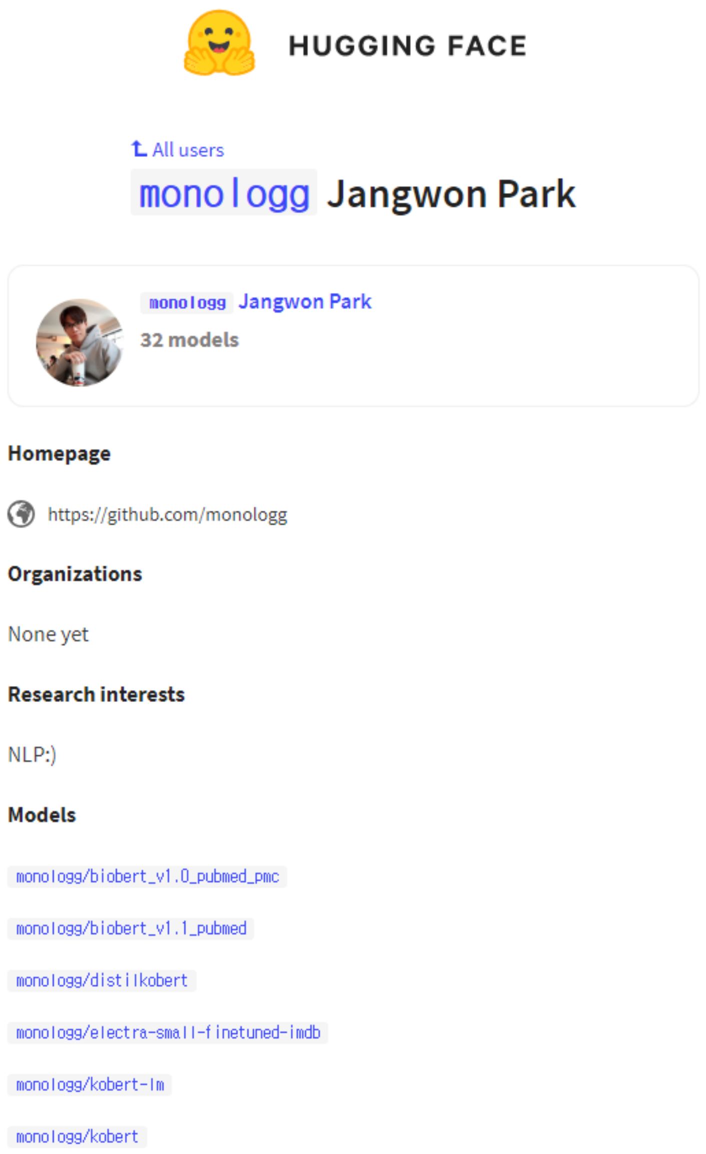
HUGGING FACE

Back to home

Community contributors 🏆

Also check out our awesome list of **organizations** 🌏.
You can join an existing org or create a new one.

mrm8488 Manuel Romero 66 models	TurkuNLP TurkuNLP Research Group university 45 models
monologg Jangwon Park 32 models	sshleifer Sam Shleifer 23 models
hfl Joint Laboratory of HIT and iFLYTEK Research (HFL) company 16 models	huseinzol05 husein zolkepli 14 models



HUGGING FACE

All users

monologg Jangwon Park

monologg Jangwon Park
32 models

Homepage

https://github.com/monologg

Organizations

None yet

Research interests

NLP:)

Models

monologg/biobert_v1.0_pubmed_pmc

monologg/biobert_v1.1_pubmed

monologg/distilkobert

monologg/electra-small-finetuned-imdb

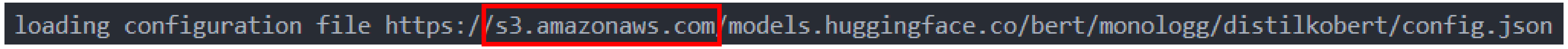
monologg/kobert-lm

monologg/kobert

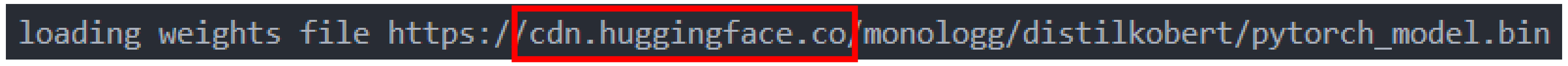
How to upload your model

- https://huggingface.co/transformers/model_sharing.html
- 일단 회원가입 진행 (https://huggingface.co/join)
- $ transformers-cli upload biobert_v1.1_pubmed

CDN issue



```
loading configuration file https://s3.amazonaws.com/models.huggingface.co/bert/monologg/distilkobert/config.json
```

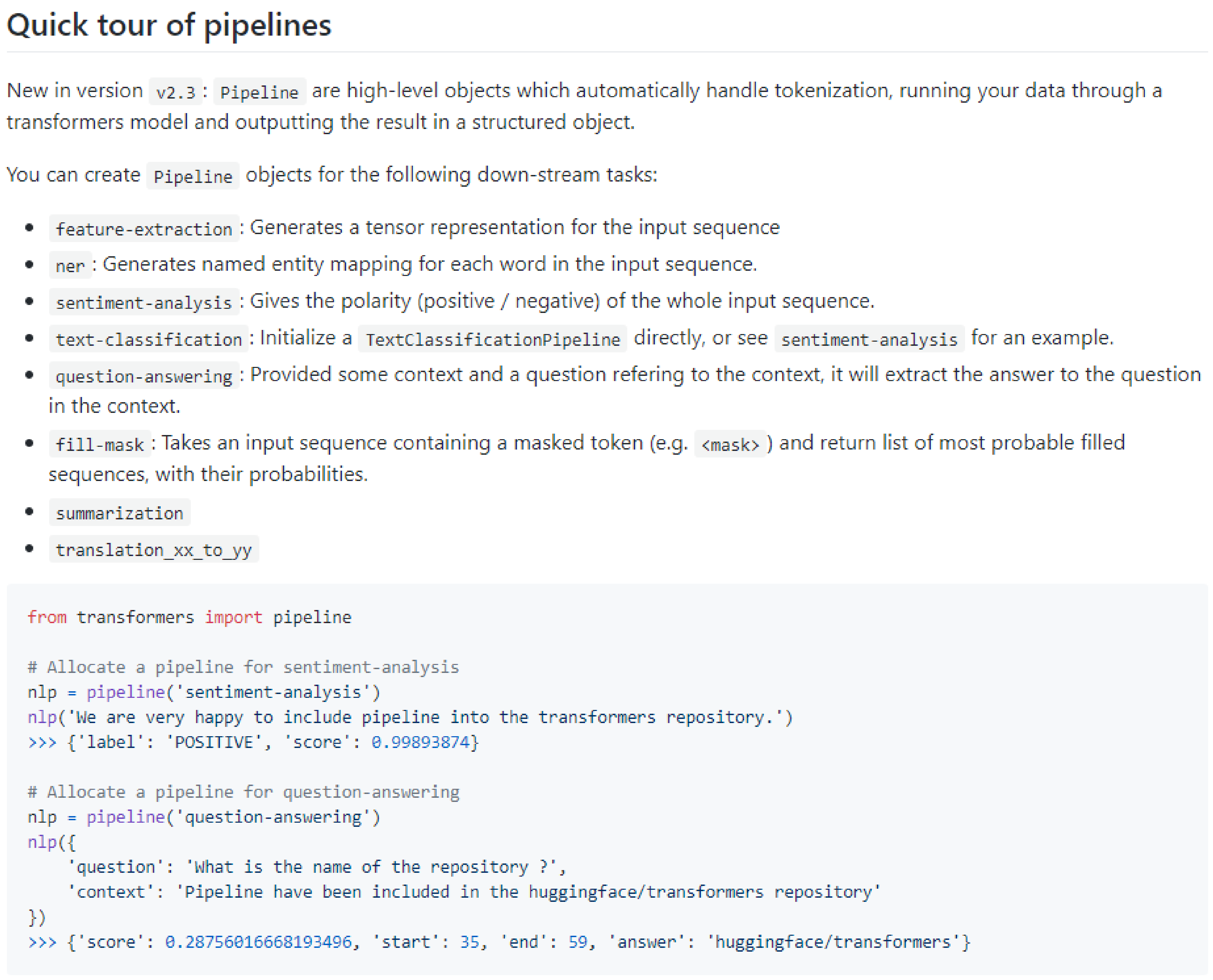


```
loading weights file https://cdn.huggingface.co/monologg/distilkobert/pytorch_model.bin
```

- model과 같이 큰 용량은 cdn(캐시)을 통하여 전송됨
- 모델을 step 별로 테스트 하는 과정에서 다시 업로드하는 경우가 있는데,

  cdn으로 인해 업데이트가 되지 않는 경우가 있음

7. Pipeline



Quick tour of pipelines

New in version `v2.3`: `Pipeline` are high-level objects which automatically handle tokenization, running your data through a transformers model and outputting the result in a structured object.

You can create `Pipeline` objects for the following down-stream tasks:

- `feature-extraction`: Generates a tensor representation for the input sequence
- `ner`: Generates named entity mapping for each word in the input sequence.
- `sentiment-analysis`: Gives the polarity (positive / negative) of the whole input sequence.
- `text-classification`: Initialize a `TextClassificationPipeline` directly, or see `sentiment-analysis` for an example.
- `question-answering`: Provided some context and a question refering to the context, it will extract the answer to the question in the context.
- `fill-mask`: Takes an input sequence containing a masked token (e.g. `<mask>`) and return list of most probable filled sequences, with their probabilities.
- `summarization`
- `translation_xx_to_yy`

```python
from transformers import pipeline

# Allocate a pipeline for sentiment-analysis
nlp = pipeline('sentiment-analysis')
nlp('We are very happy to include pipeline into the transformers repository.')
>>> {'label': 'POSITIVE', 'score': 0.99893874}

# Allocate a pipeline for question-answering
nlp = pipeline('question-answering')
nlp({
    'question': 'What is the name of the repository ?',
    'context': 'Pipeline have been included in the huggingface/transformers repository'
})
>>> {'score': 0.28756016668193496, 'start': 35, 'end': 59, 'answer': 'huggingface/transformers'}
```

1. NSMC

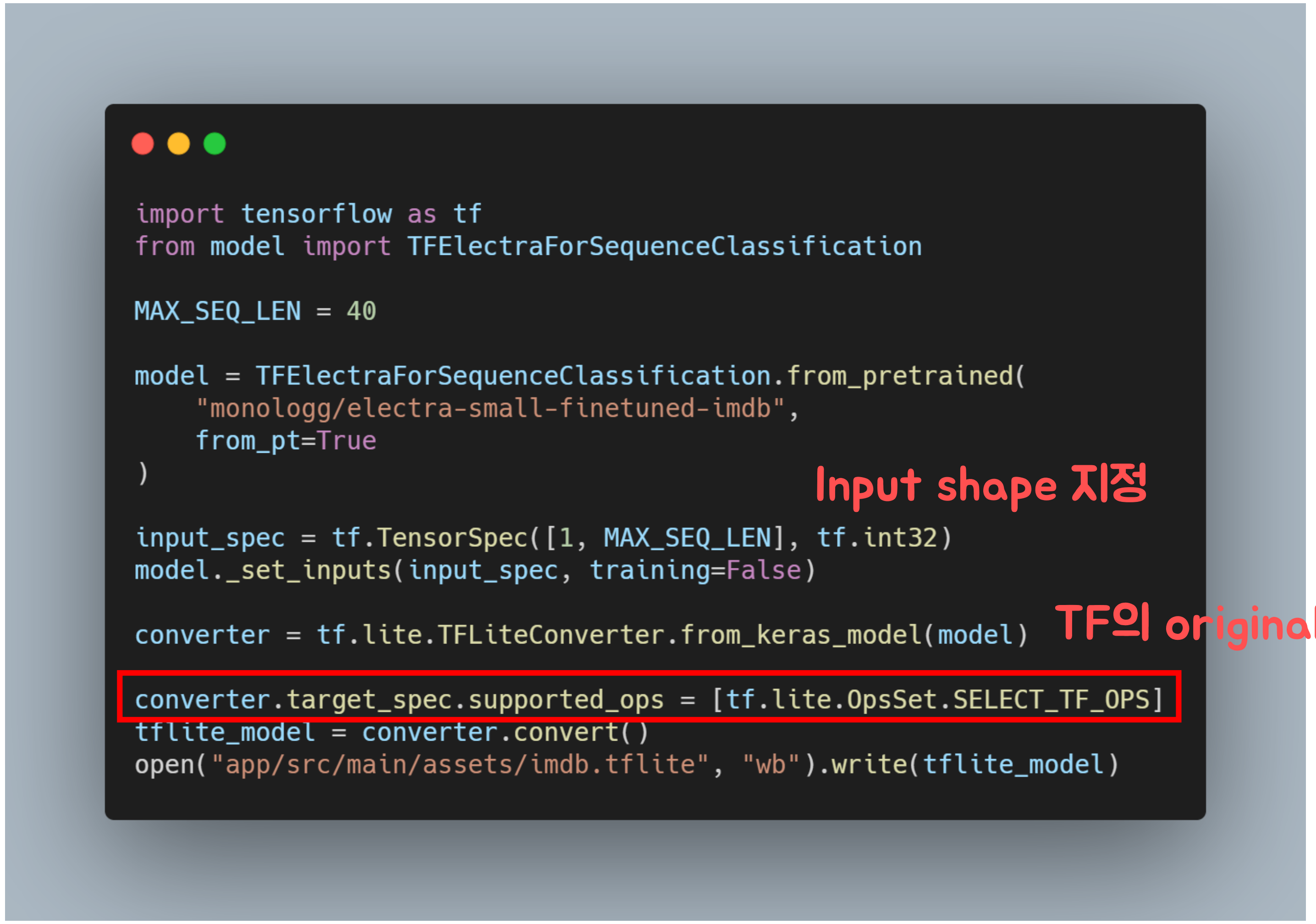
```python
from transformers import ElectraTokenizer, pipeline
from model import ElectraForSequenceClassification

tokenizer = ElectraTokenizer.from_pretrained("monologg/koelectra-small-finetuned-sentiment")
model = ElectraForSequenceClassification.from_pretrained("monologg/koelectra-small-finetuned-sentiment")

nsmc = pipeline(
    "sentiment-analysis",
    tokenizer=tokenizer,
    model=model
)

print(nsmc("이 영화는 미쳤다. 넷플릭스가 일상화된 시대에 극장이 존재해야하는 이유를 증명해준다."))

# Out
[{'label': 'positive', 'score': 0.8636718392372131}]
```

- https://github.com/monologg/KoELECTRA-Pipeline
- But...그대로 가져다 쓰기에는 불편한 부분이 없지 않다....

8. On-Device

주의! 뇌피셜이 (매우) 많을 수 있음

1. TFLite



```python
import tensorflow as tf
from model import TFElectraForSequenceClassification

MAX_SEQ_LEN = 40

model = TFElectraForSequenceClassification.from_pretrained(
    "monologg/electra-small-finetuned-imdb",
    from_pt=True
)

input_spec = tf.TensorSpec([1, MAX_SEQ_LEN], tf.int32)
model._set_inputs(input_spec, training=False)

converter = tf.lite.TFLiteConverter.from_keras_model(model)

converter.target_spec.supported_ops = [tf.lite.OpsSet.SELECT_TF_OPS]
tflite_model = converter.convert()
open("app/src/main/assets/imdb.tflite", "wb").write(tflite_model)
```

Input shape 지정

TF의 original ops로 변환

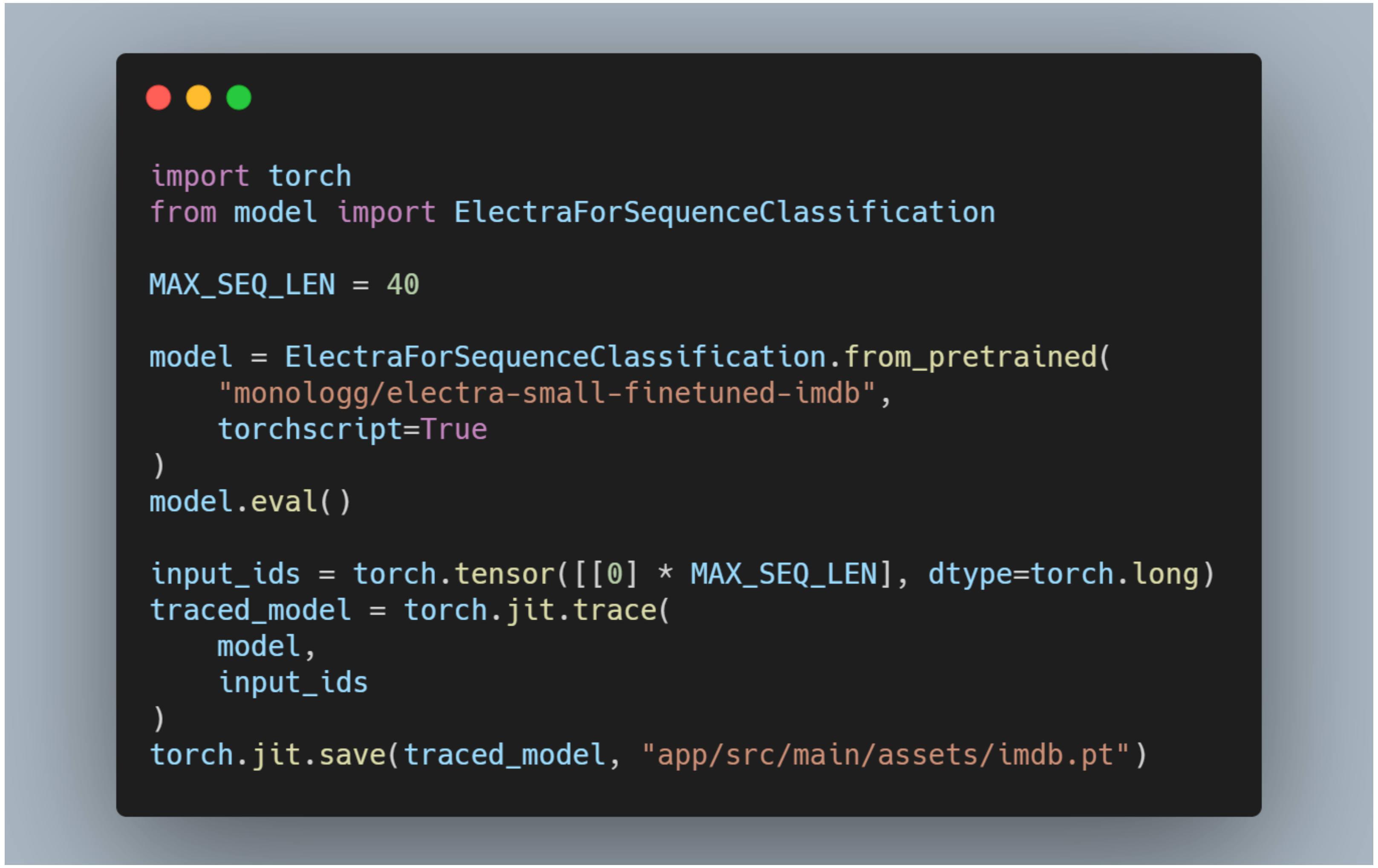
TFLite 변환 시 Issue

- Tflite용 ops로 변환할 수 없음
  - Embedding 쪽의 gather 함수에서 이슈가 있음
  - https://www.tensorflow.org/lite/guide/ops_select
  - 그래서 변환해도 사이즈가 똑같음

TFLite 변환 시 Issue

- TF v2.1.0 사용 권장 (v2.2.0 에서 버그 발생)
- GPU에서 변환해야 함 (CPU로 하면 변환 X)
- fp16 convert 도 가능
- 고정된 길이만 입력받을 수 있음!
    - 길이가 8이여도 MAX_SEQ_LEN=40으로 넣어야 함

2. Torchscript



```
import torch
from model import ElectraForSequenceClassification

MAX_SEQ_LEN = 40

model = ElectraForSequenceClassification.from_pretrained(
    "monologg/electra-small-finetuned-imdb",
    torchscript=True
)
model.eval()

input_ids = torch.tensor([[0] * MAX_SEQ_LEN], dtype=torch.long)
traced_model = torch.jit.trace(
    model,
    input_ids
)
torch.jit.save(traced_model, "app/src/main/assets/imdb.pt")
```

Torchscript 특징, 이슈

- https://huggingface.co/transformers/torchscript.html
- 문서상으로는 고정길이만 허용되는 거 같아 보여도 가변 길이도 허용됨
- 첫 forward에서 굉장히 느림 (preheating work)
- batch_size=1일 시 오히려 torchscript가 느리다는 얘기들이 있음
- cpu에서는 torchscript의 속도 향상이 크지 않음
  - https://medium.com/huggingface/benchmarking-transformers-pytorch-and-tensorflow-e2917fb891c2

안드로이드 데모 시연

https://github.com/monologg/transformers-android-demo

9. Transformers + TPU

Pytorch와 TPU를 같이 쓰는 것은 비추

미지원 연산을 마주치는 순간 엄청 느려짐

https://github.com/pytorch/xla

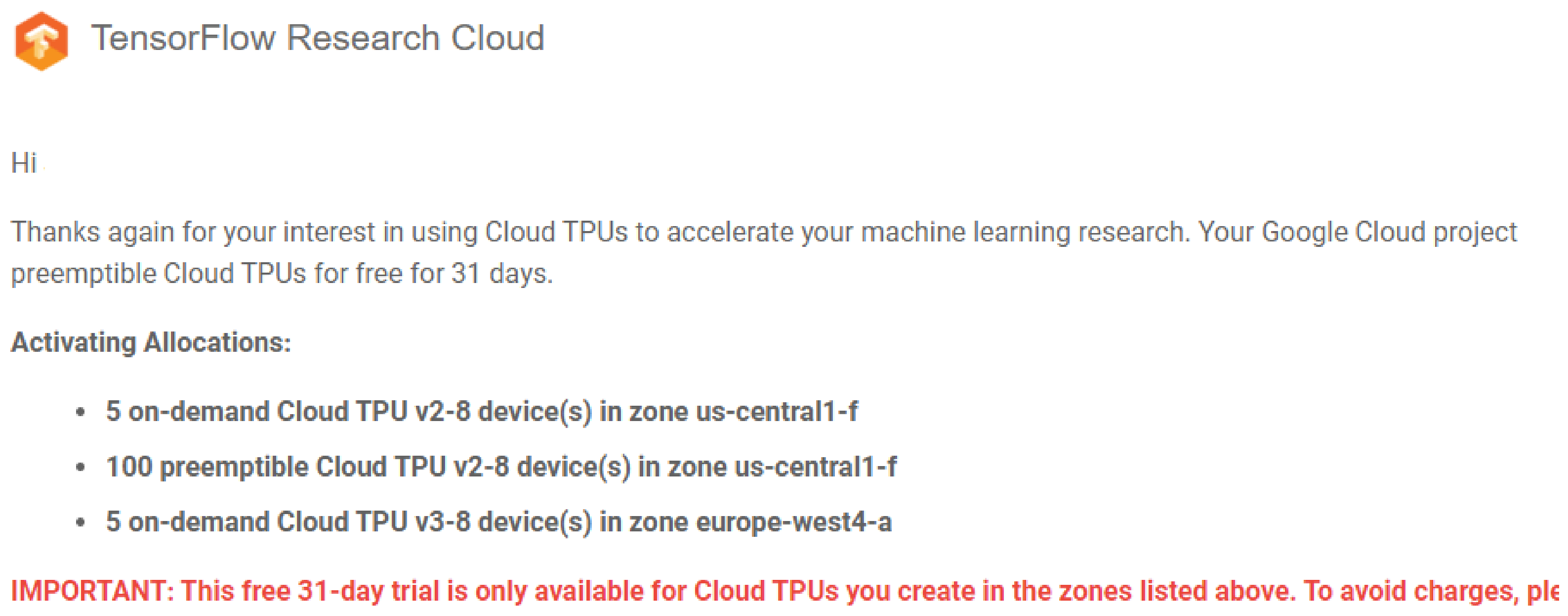
TFRC (Tensorflow Research Cloud)

TensorFlow Research Cloud

무료 Cloud TPU로 최첨단 머신러닝 연구를 가속화하세요.

-> 우리 (구글) 사장님이 미쳤어요!

TFRC (Tensorflow Research Cloud)


TensorFlow Research Cloud

Hi

Thanks again for your interest in using Cloud TPUs to accelerate your machine learning research. Your Google Cloud project preemptible Cloud TPUs for free for 31 days.

Activating Allocations:

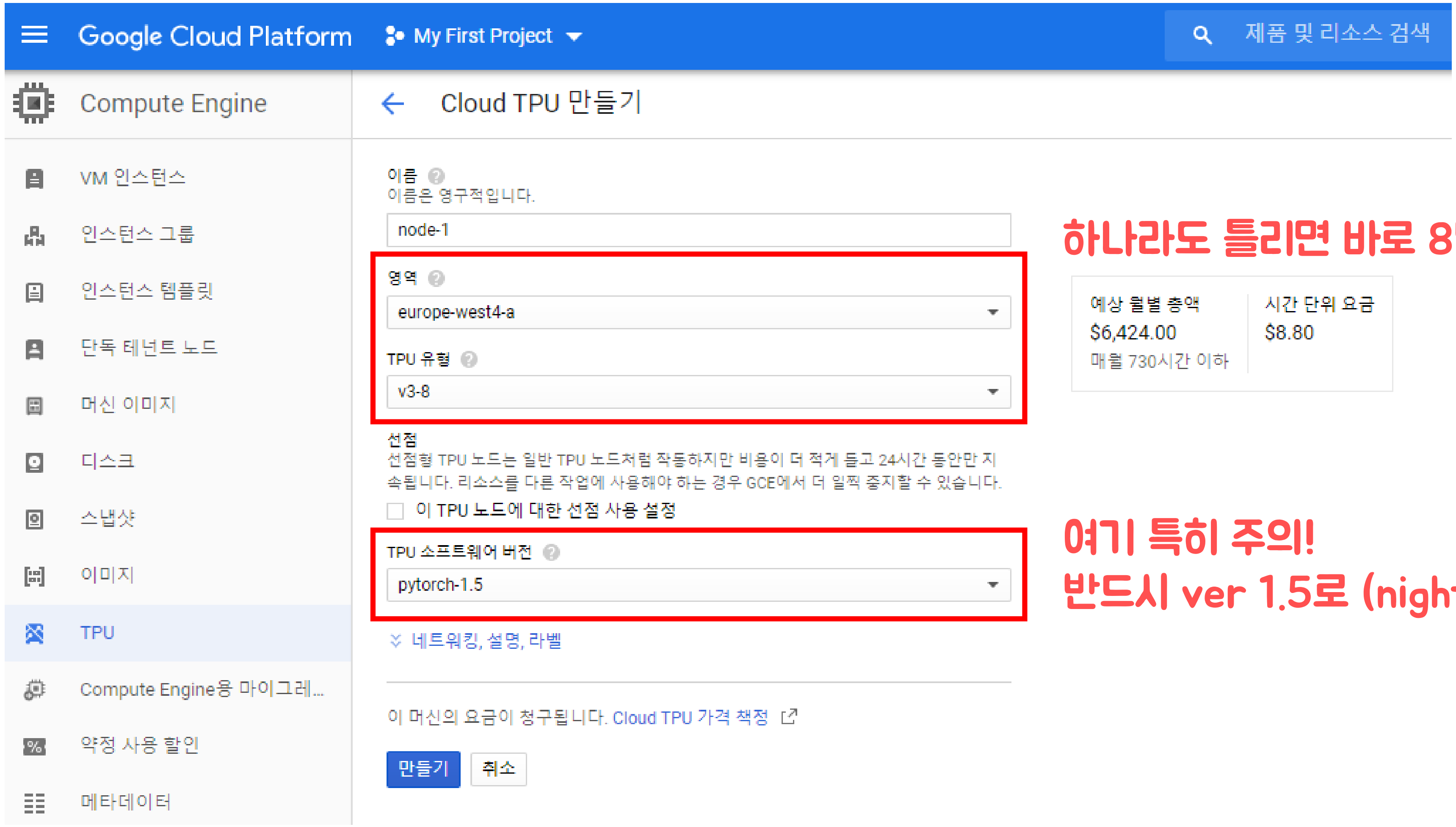
- **5 on-demand Cloud TPU v2-8 device(s) in zone us-central1-f**
- **100 preemptible Cloud TPU v2-8 device(s) in zone us-central1-f**
- **5 on-demand Cloud TPU v3-8 device(s) in zone europe-west4-a**

IMPORTANT: This free 31-day trial is only available for Cloud TPUs you create in the zones listed above. To avoid charges, ple

- v2-8 5개 + v3-8 5개
- 1시간에 8달러...ㄷㄷ

GCP 세팅

1. TPU 생성



Cloud TPU 만들기

이름
이름은 영구적입니다.
node-1

영역
europe-west4-a

TPU 유형
v3-8

선점
선점형 TPU 노드는 일반 TPU 노드처럼 작동하지만 비용이 더 적게 들고 24시간 동안만 지속됩니다. 리소스를 다른 작업에 사용해야 하는 경우 GCE에서 더 일찍 중지할 수 있습니다.

- [ ] 이 TPU 노드에 대한 선점 사용 설정

TPU 소프트웨어 버전
pytorch-1.5

네트워킹, 설명, 라벨

이 머신의 요금이 청구됩니다. Cloud TPU 가격 책정

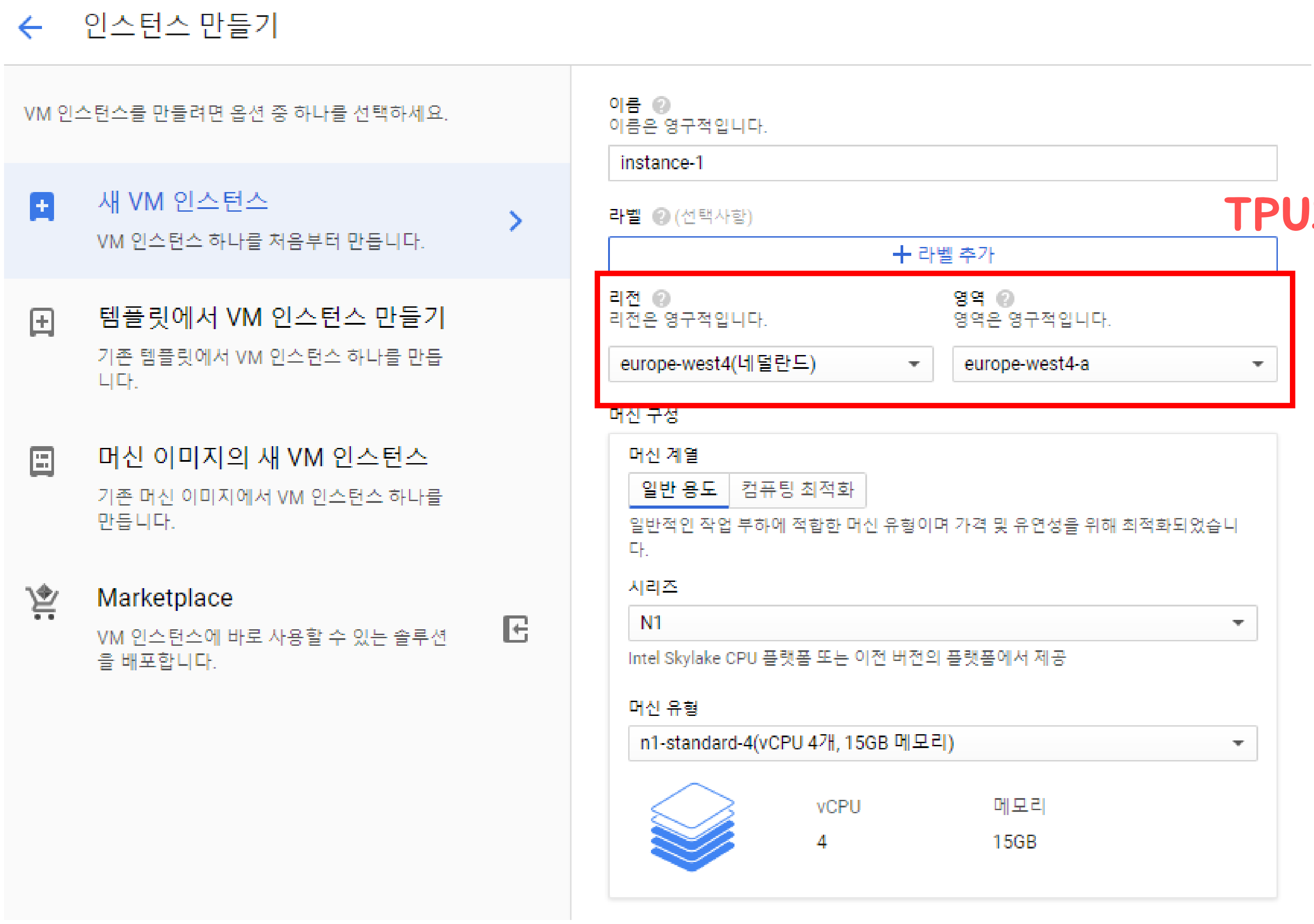
하나라도 틀리면 바로 8달러...

예상 월별 총액	시간 단위 요금
$6,424.00	$8.80
매월 730시간 이하	

여기 특히 주의!
반드시 ver 1.5로 (nightly 비추)

GCP 세팅

2. Instance 생성



인스턴스 만들기

VM 인스턴스를 만들려면 옵션 중 하나를 선택하세요.

새 VM 인스턴스
VM 인스턴스 하나를 처음부터 만듭니다.

템플릿에서 VM 인스턴스 만들기
기존 템플릿에서 VM 인스턴스 하나를 만듭니다.

머신 이미지의 새 VM 인스턴스
기존 머신 이미지에서 VM 인스턴스 하나를 만듭니다.

Marketplace
VM 인스턴스에 바로 사용할 수 있는 솔루션을 배포합니다.

이름
이름은 영구적입니다.
instance-1

라벨 (선택사항)
+ 라벨 추가

리전
리전은 영구적입니다.
europe-west4(네덜란드)

영역
영역은 영구적입니다.
europe-west4-a

머신 구성

머신 계열
일반 용도 | 컴퓨팅 최적화
일반적인 작업 부하에 적합한 머신 유형이며 가격 및 유연성을 위해 최적화되었습니다.

시리즈
N1
Intel Skylake CPU 플랫폼 또는 이전 버전의 플랫폼에서 제공

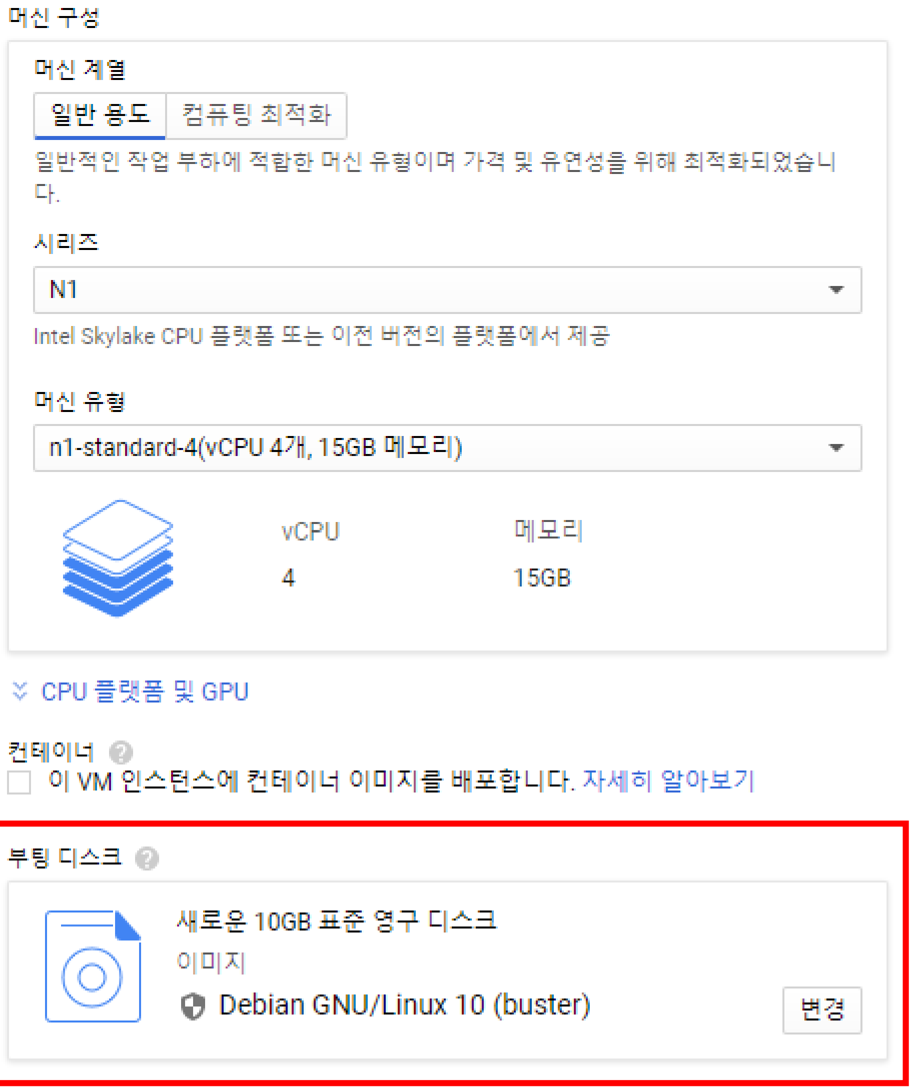
머신 유형
n1-standard-4(vCPU 4개, 15GB 메모리)

vCPU	메모리
4	15GB

TPU와 동일 Region이어야 함

GCP 세팅

2. Instance 생성



머신 구성

머신 계열

일반 용도 | 컴퓨팅 최적화

일반적인 작업 부하에 적합한 머신 유형이며 가격 및 유연성을 위해 최적화되었습니다.

시리즈

N1

Intel Skylake CPU 플랫폼 또는 이전 버전의 플랫폼에서 제공

머신 유형

n1-standard-4(vCPU 4개, 15GB 메모리)

vCPU	메모리
4	15GB

CPU 플랫폼 및 GPU

컨테이너

이 VM 인스턴스에 컨테이너 이미지를 배포합니다. 자세히 알아보기

부팅 디스크

새로운 10GB 표준 영구 디스크

이미지

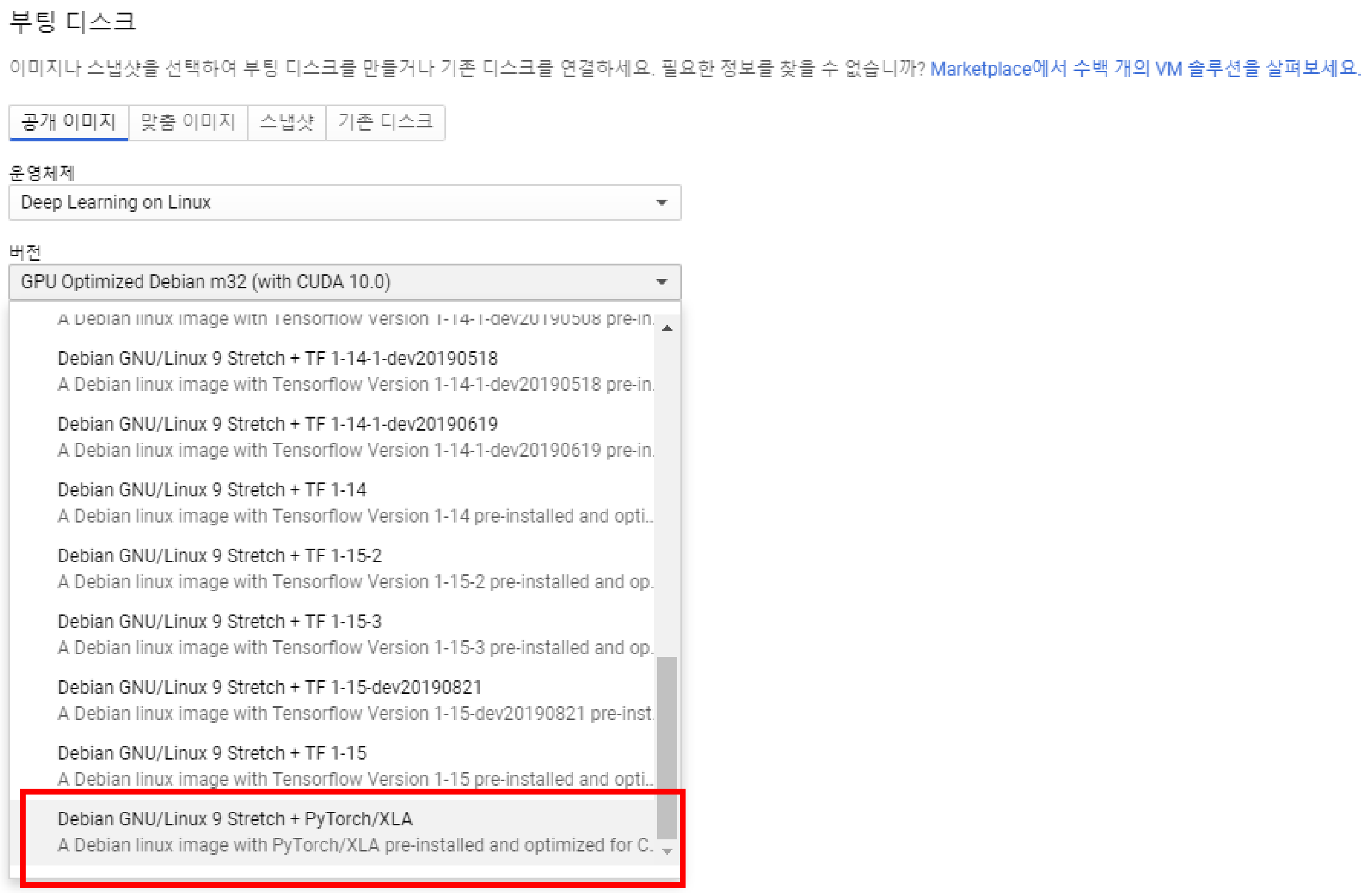
Debian GNU/Linux 10 (buster)

변경

이미지 변경 필요!

GCP 세팅

2. Instance 생성



부팅 디스크

이미지나 스냅샷을 선택하여 부팅 디스크를 만들거나 기존 디스크를 연결하세요. 필요한 정보를 찾을 수 없습니까? Marketplace에서 수백 개의 VM 솔루션을 살펴보세요.

운영체제

Deep Learning on Linux

버전

GPU Optimized Debian m32 (with CUDA 10.0)

A Debian linux image with Tensorflow Version 1-14-1-dev20190508 pre-in.

Debian GNU/Linux 9 Stretch + TF 1-14-1-dev20190518
A Debian linux image with Tensorflow Version 1-14-1-dev20190518 pre-in.

Debian GNU/Linux 9 Stretch + TF 1-14-1-dev20190619
A Debian linux image with Tensorflow Version 1-14-1-dev20190619 pre-in.

Debian GNU/Linux 9 Stretch + TF 1-14
A Debian linux image with Tensorflow Version 1-14 pre-installed and opti..

Debian GNU/Linux 9 Stretch + TF 1-15-2
A Debian linux image with Tensorflow Version 1-15-2 pre-installed and op.

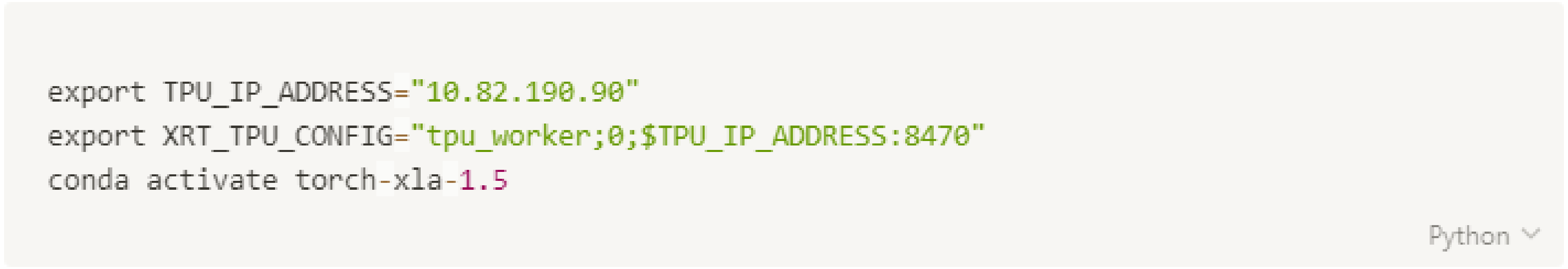
Debian GNU/Linux 9 Stretch + TF 1-15-3
A Debian linux image with Tensorflow Version 1-15-3 pre-installed and op.

Debian GNU/Linux 9 Stretch + TF 1-15-dev20190821
A Debian linux image with Tensorflow Version 1-15-dev20190821 pre-inst.

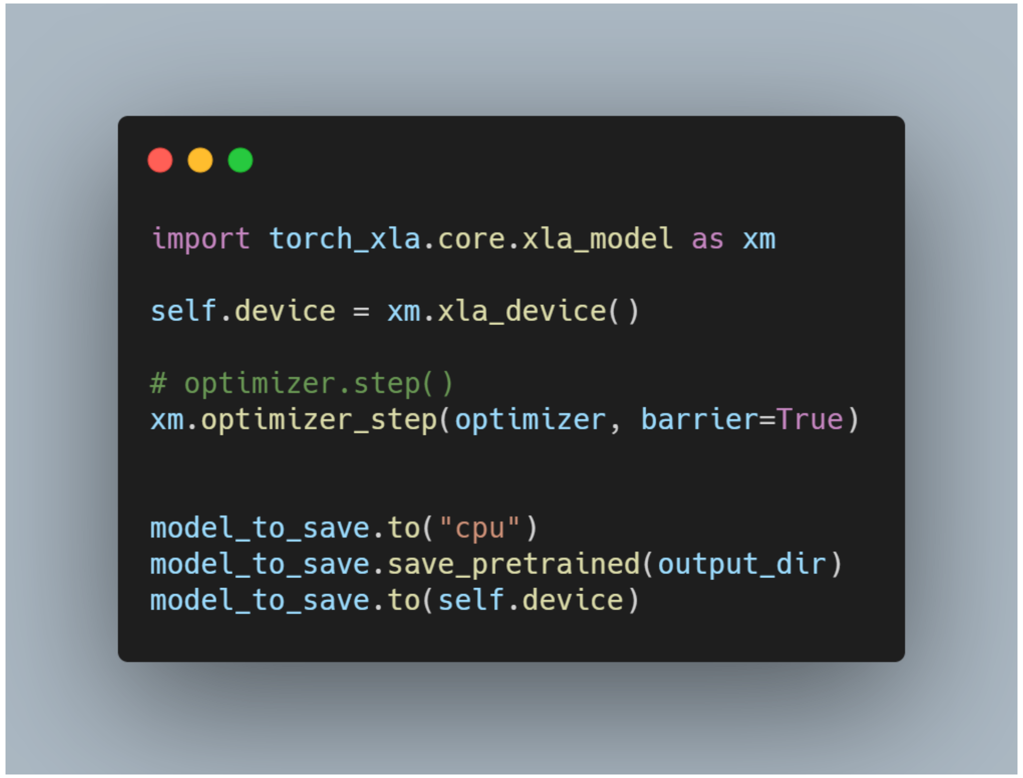
Debian GNU/Linux 9 Stretch + TF 1-15
A Debian linux image with Tensorflow Version 1-15 pre-installed and opti..

Debian GNU/Linux 9 Stretch + PyTorch/XLA
A Debian linux image with PyTorch/XLA pre-installed and optimized for C.

환경변수 세팅


```
export TPU_IP_ADDRESS="10.82.190.90"
export XRT_TPU_CONFIG="tpu_worker;0;$TPU_IP_ADDRESS:8470"
conda activate torch-xla-1.5
```

8코어 중 1개만 사용하는 코드



```python
import torch_xla.core.xla_model as xm

self.device = xm.xla_device()

# optimizer.step()
xm.optimizer_step(optimizer, barrier=True)


model_to_save.to("cpu")
model_to_save.save_pretrained(output_dir)
model_to_save.to(self.device)
```

1. TPU Device로 변경
2. optimzer step 변경
3. 모델 저장시 CPU로 잠깐 옮기기

https://github.com/pytorch/xla/blob/master/API_GUIDE.md

8코어 전부 사용하는 코드

https://github.com/huggingface/transformers/blob/700ccf6e35/examples/run_tpu_glue.py

- 이거보다 좋은 레퍼런스 코드를 찾기 어려움 (by Pytorch TPU Contributor)
- rendevzous 포인트가 핵심 (multi-gpu 코드랑 일맥상통하는 부분이 있음)

여전히 버그가 없는 건 아니다.

하지만 수많은 contributor들이 고쳐나가고 있다!

감사합니다!

2020.06.03 박장원