Large Language Model (LLM) with Python

Faranaksadat Solat

intelligent Mobile Edge Computing Systems Lab (iMES Lab.)

Department of AI Software

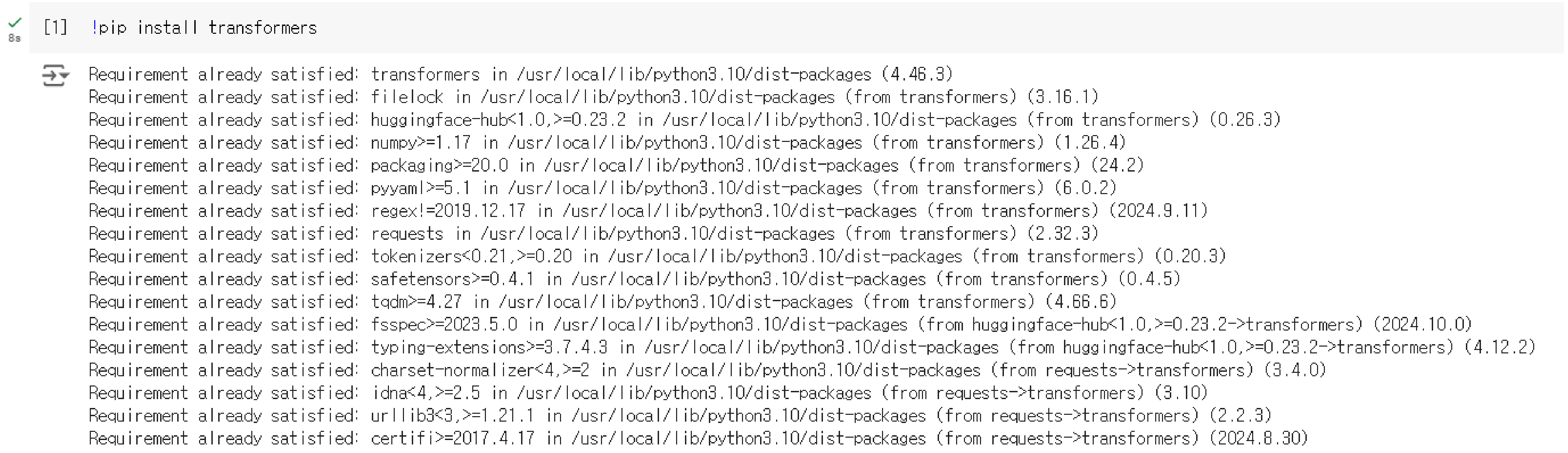
Gachon University

December 2024

Table of Contents

Starting point

- Using Google Colab
- Change the run time to T4 GPU (16 cores)
- Install transformers

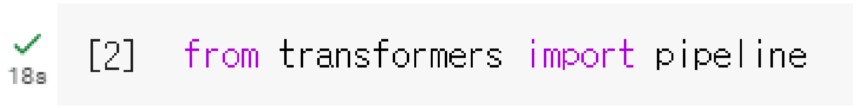


```
[1]  !pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.46.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.26.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.21,>=0.20 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.20.3)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.6)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (2024.10.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.8.30)
```

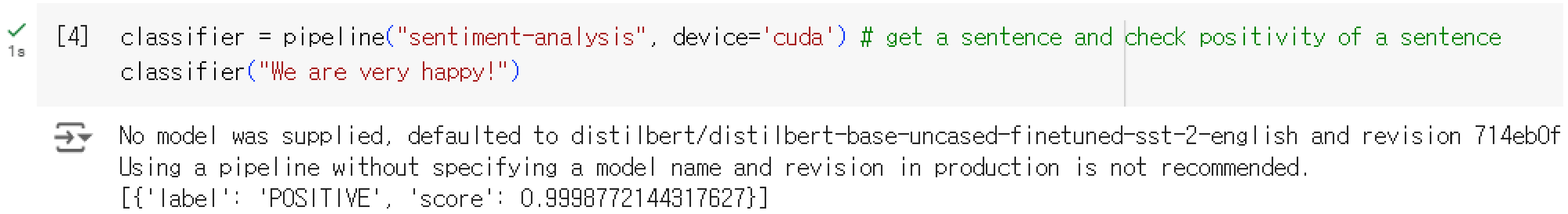
A classification model

- Import pipeline from transformers



```
[2]  from transformers import pipeline
```

- Using *sentiment-analysis* for checking the positivity of a sentence



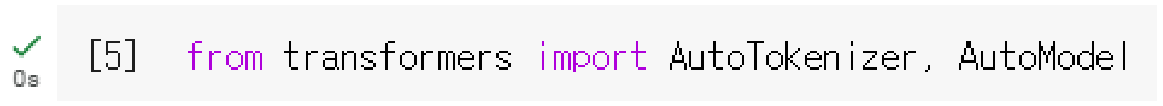
```
[4]  classifier = pipeline("sentiment-analysis", device='cuda') # get a sentence and check positivity of a sentence
     classifier("We are very happy!")
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f
Using a pipeline without specifying a model name and revision in production is not recommended.
[{'label': 'POSITIVE', 'score': 0.9998772144317627}]
```

- How it works?
- We tokenize the text and make a dictionary
- Then, we train in our model

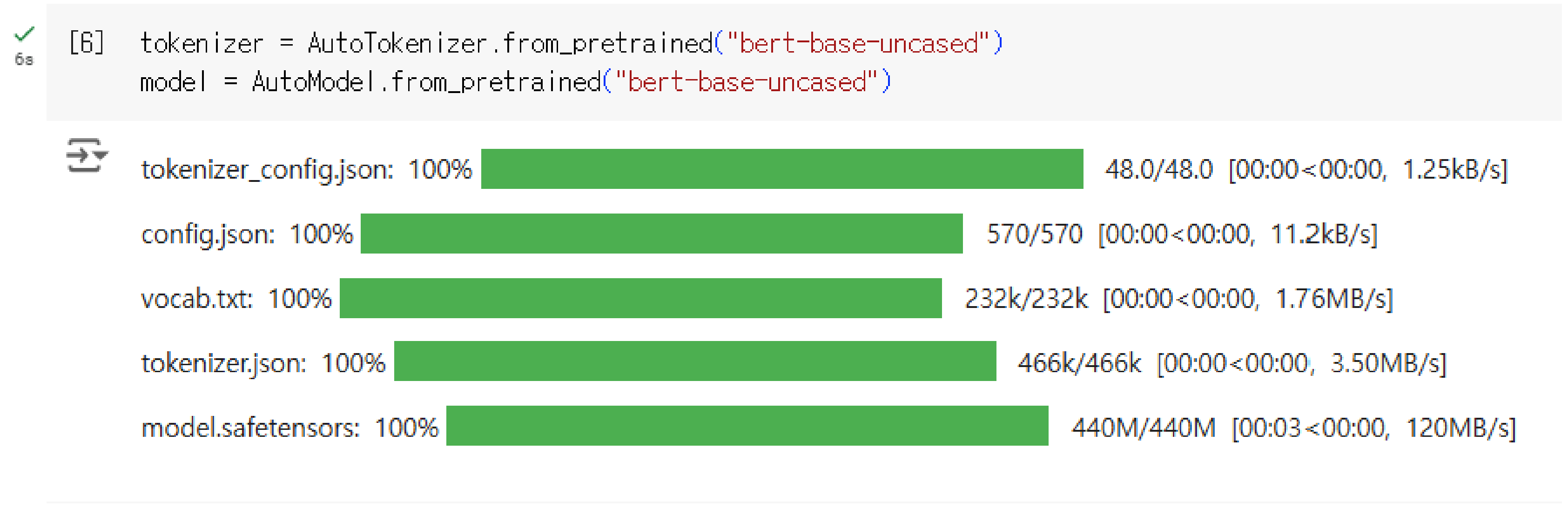
Bert model

- Bert get a content and convert it into a vector
- Then we import AutoTokenizer and AutoModel



```
[5]  from transformers import AutoTokenizer, AutoModel
```

- We define tokenizer and model



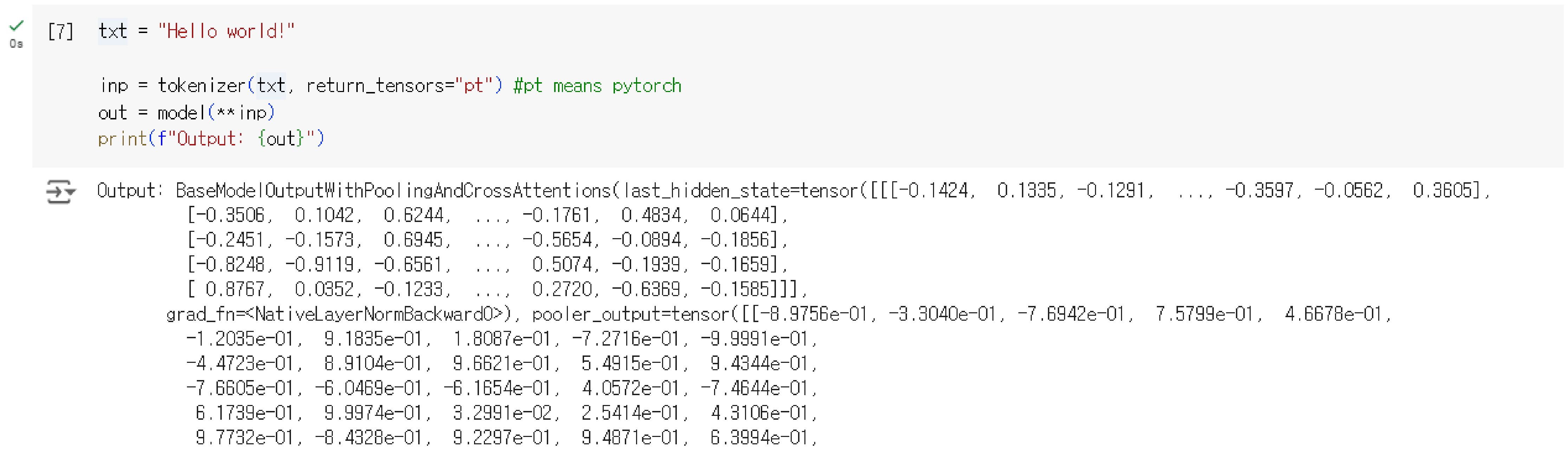
```
[6]  tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
     model = AutoModel.from_pretrained("bert-base-uncased")
```

```
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 1.25kB/s]
config.json: 100%  570/570 [00:00<00:00, 11.2kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 1.76MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 3.50MB/s]
model.safetensors: 100%  440M/440M [00:03<00:00, 120MB/s]
```

- Hint: The pretrained model for tokenizer and the model should be the same

Bert model (Pytorch)

- We prepare the out put for pytorch (pt)



```
[7]  txt = "Hello world!"

     inp = tokenizer(txt, return_tensors="pt") #pt means pytorch
     out = model(**inp)
     print(f"Output: {out}")
```

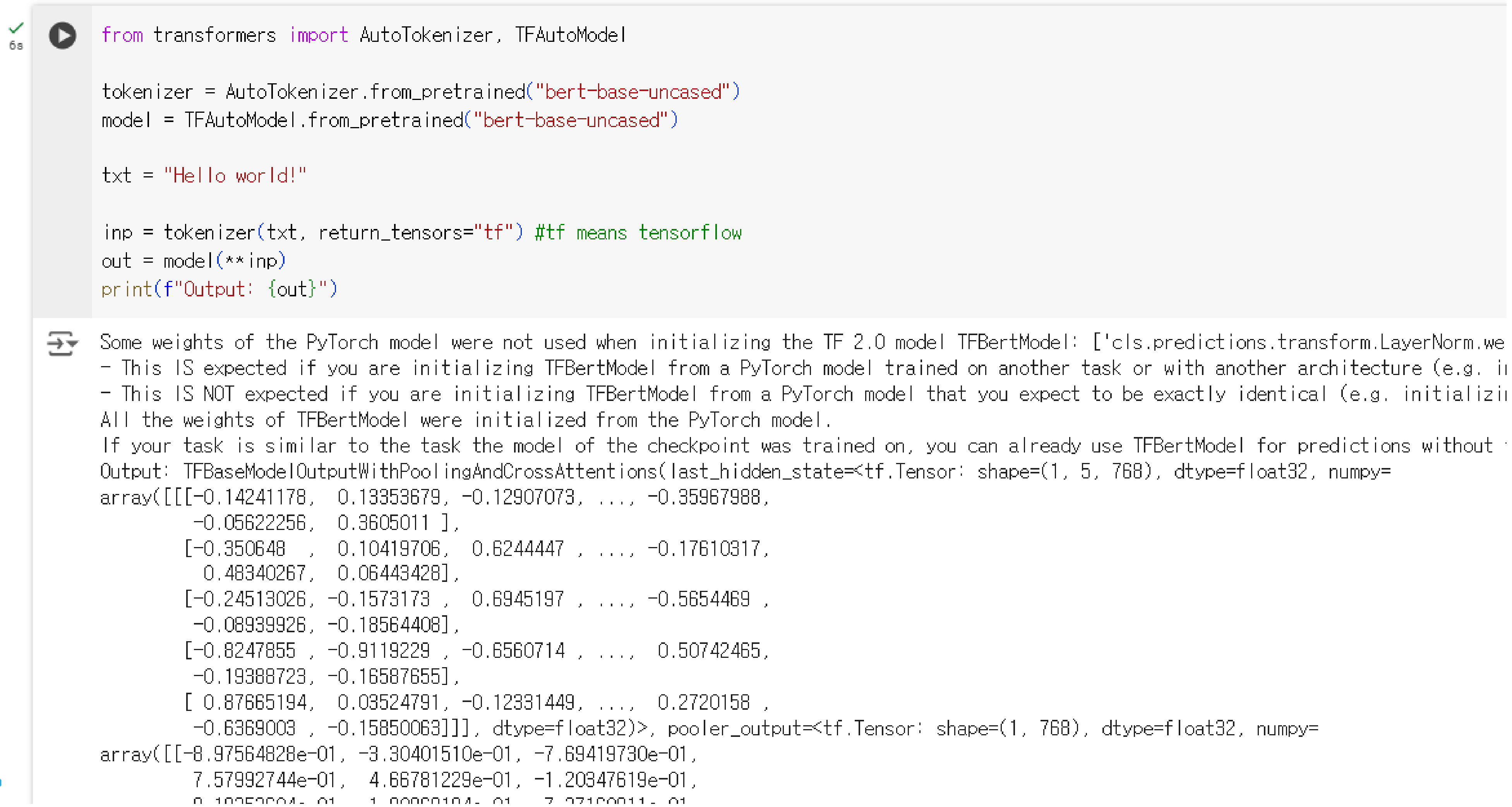
```
Output: BaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=tensor([[[-0.1424,  0.1335, -0.1291,  ..., -0.3597, -0.0562,  0.3605],
         [-0.3506,  0.1042,  0.6244,  ..., -0.1761,  0.4834,  0.0644],
         [-0.2451, -0.1573,  0.6945,  ..., -0.5654, -0.0894, -0.1856],
         [-0.8248, -0.9119, -0.6561,  ...,  0.5074, -0.1939, -0.1659],
         [ 0.8767,  0.0352, -0.1233,  ...,  0.2720, -0.6369, -0.1585]]],
       grad_fn=<NativeLayerNormBackward0>), pooler_output=tensor([[-8.9756e-01, -3.3040e-01, -7.6942e-01,  7.5799e-01,  4.6678e-01,
         -1.2035e-01,  9.1835e-01,  1.8087e-01, -7.2716e-01, -9.9991e-01,
         -4.4723e-01,  8.9104e-01,  9.6621e-01,  5.4915e-01,  9.4344e-01,
         -7.6605e-01, -6.0469e-01, -6.1654e-01,  4.0572e-01, -7.4644e-01,
          6.1739e-01,  9.9974e-01,  3.2991e-02,  2.5414e-01,  4.3106e-01,
          9.7732e-01, -8.4328e-01,  9.2297e-01,  9.4871e-01,  6.3994e-01,
```

- Question: Do you know why in output we put **?

  Answer: Because the input is a dictionary and we must open each field

Bert model (Tensorflow)

- We prepare the out put for Tensorflow (tf)

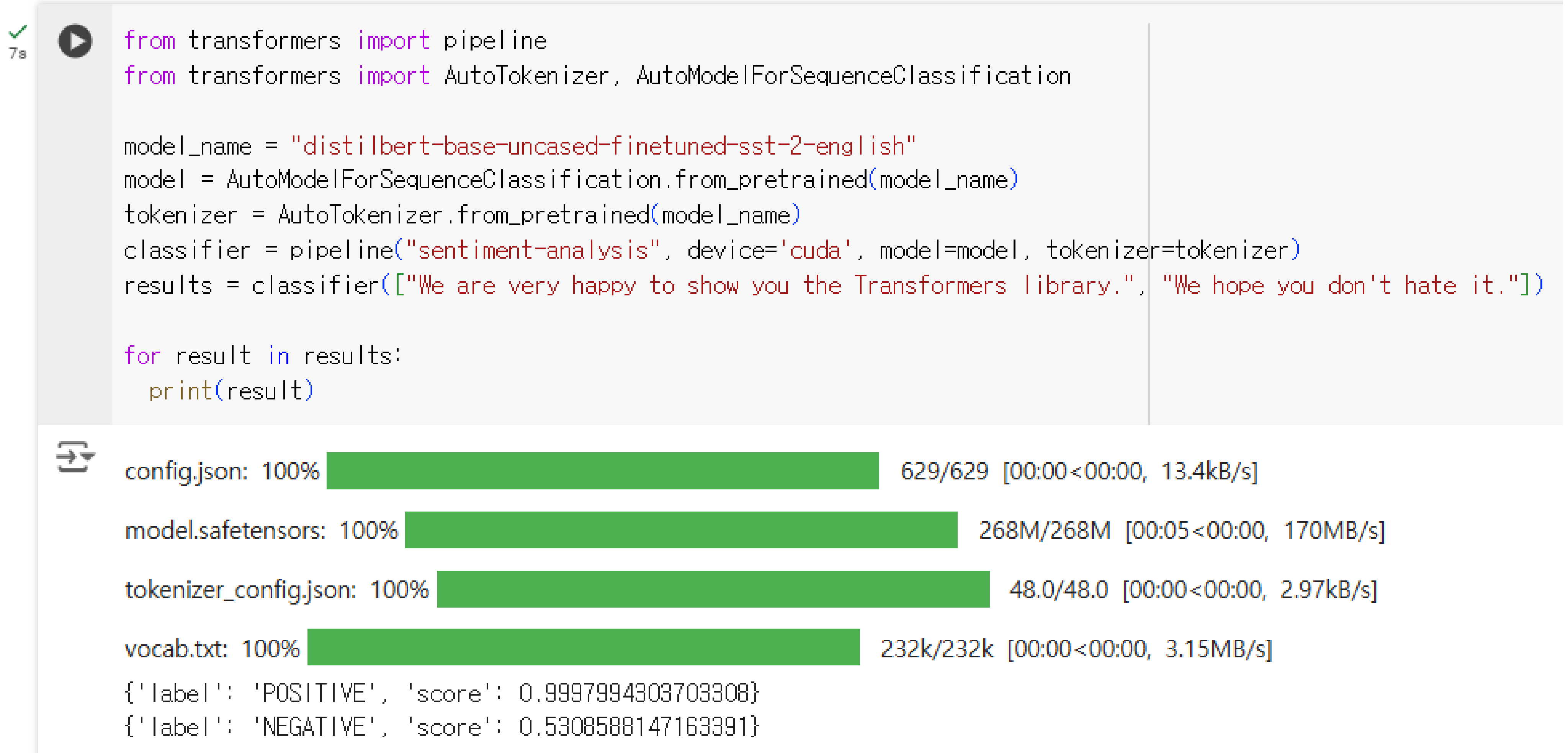


```
from transformers import AutoTokenizer, TFAutoModel

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = TFAutoModel.from_pretrained("bert-base-uncased")

txt = "Hello world!"

inp = tokenizer(txt, return_tensors="tf") #tf means tensorflow
out = model(**inp)
print(f"Output: {out}")
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.predictions.transform.LayerNorm.we
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e.g. i
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initializi
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions without
Output: TFBaseModelOutputWithPoolingAndCrossAttentions(last_hidden_state=<tf.Tensor: shape=(1, 5, 768), dtype=float32, numpy=
array([[[-0.14241178,  0.13353679, -0.12907073, ..., -0.35967988,
         -0.05622256,  0.3605011 ],
        [-0.350648  ,  0.10419706,  0.6244447 , ..., -0.17610317,
          0.48340267,  0.06443428],
        [-0.24513026, -0.1573173 ,  0.6945197 , ..., -0.5654469 ,
         -0.08939926, -0.18564408],
        [-0.8247855 , -0.9119229 , -0.6560714 , ...,  0.50742465,
         -0.19388723, -0.16587655],
        [ 0.87665194,  0.03524791, -0.12331449, ...,  0.2720158 ,
         -0.6369003 , -0.15850063]]], dtype=float32)>, pooler_output=<tf.Tensor: shape=(1, 768), dtype=float32, numpy=
array([[-8.97564828e-01, -3.30401510e-01, -7.69419730e-01,
         7.57992744e-01,  4.66781229e-01, -1.20347619e-01,
```

Sentiment Analysis using our model

- Now we are going to use sentiment-analysis task but instead of system model we generate our model



```
from transformers import pipeline
from transformers import AutoTokenizer, AutoModelForSequenceClassification

model_name = "distilbert-base-uncased-finetuned-sst-2-english"
model = AutoModelForSequenceClassification.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
classifier = pipeline("sentiment-analysis", device='cuda', model=model, tokenizer=tokenizer)
results = classifier(["We are very happy to show you the Transformers library.", "We hope you don't hate it."])

for result in results:
  print(result)
```

```
config.json: 100%  629/629 [00:00<00:00, 13.4kB/s]
model.safetensors: 100%  268M/268M [00:05<00:00, 170MB/s]
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 2.97kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 3.15MB/s]
{'label': 'POSITIVE', 'score': 0.9997994303703308}
{'label': 'NEGATIVE', 'score': 0.5308588147163391}
```

Sentiment Analysis using our model

“We are very happy to show you the Transformers library.”

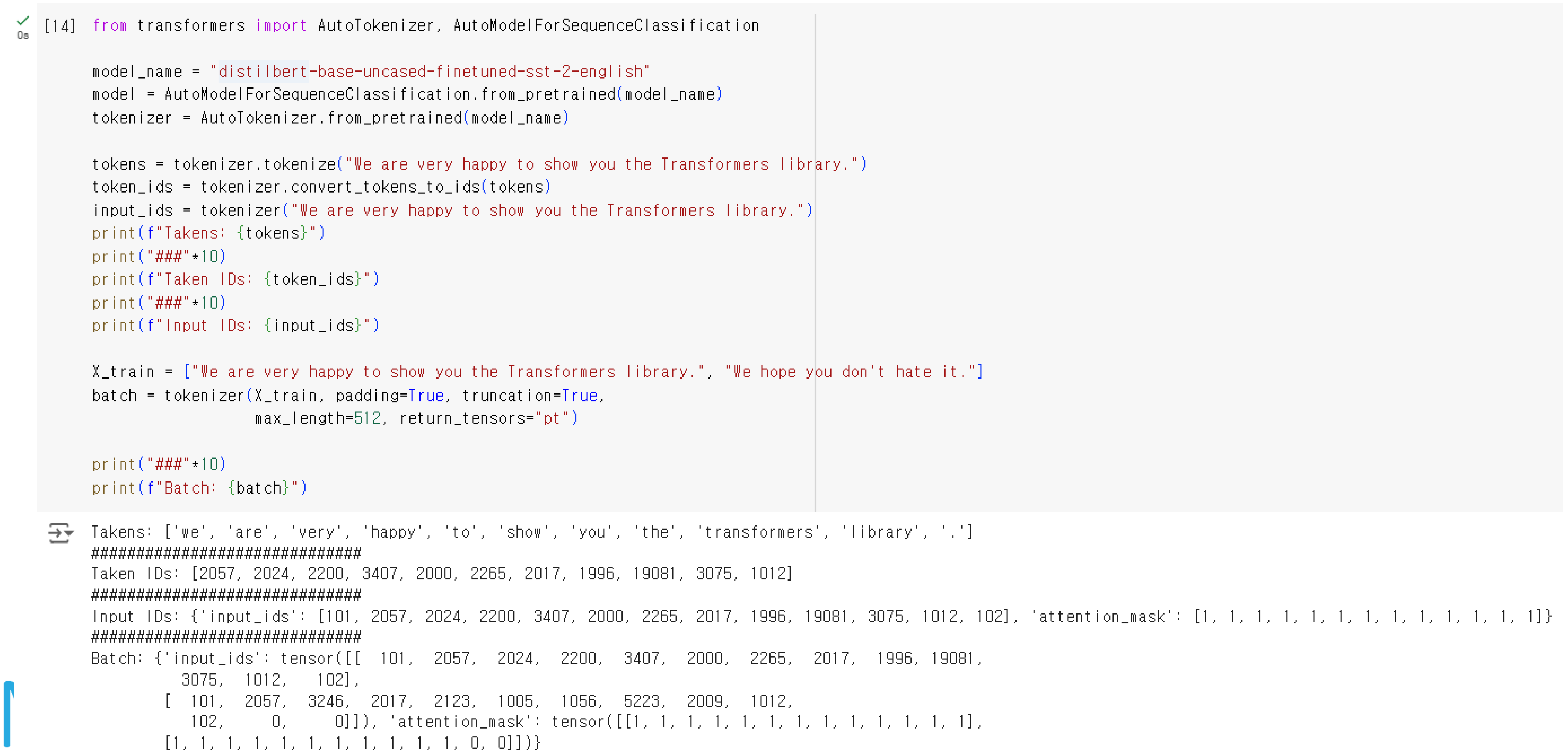
```
{'label': 'POSITIVE', 'score': 0.9997994303703308}
```

"We hope you don't hate it.“

```
{'label': 'NEGATIVE', 'score': 0.5308588147163391}
```

Sentiment Analysis using our model (Tokenizer)

- Now we are going to use sentiment-analysis task but instead of system model we generate our model

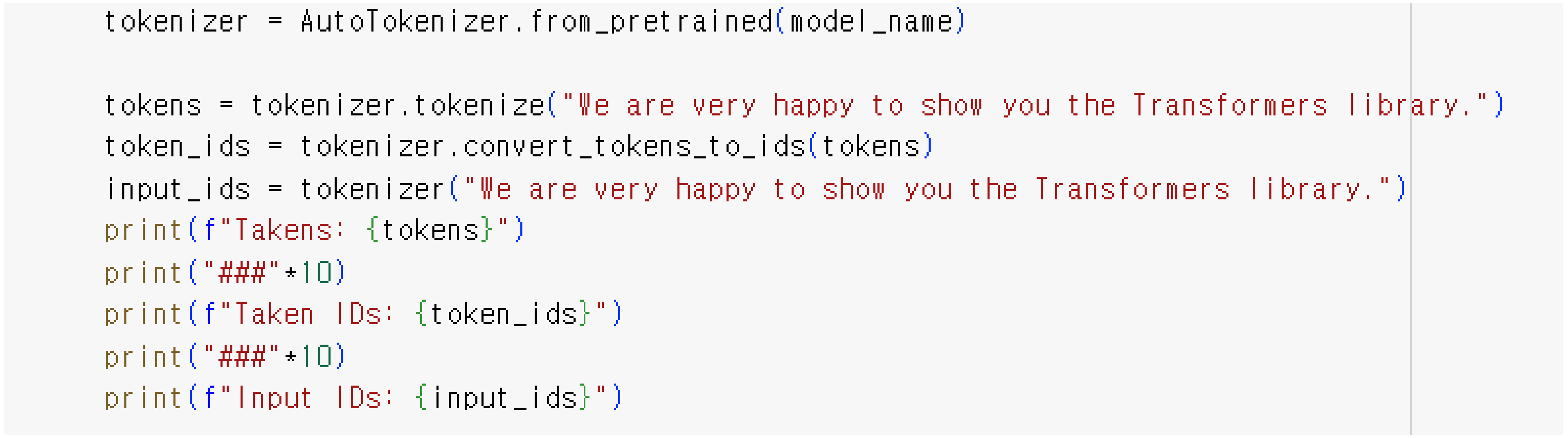


```
[14] from transformers import AutoTokenizer, AutoModelForSequenceClassification

     model_name = "distilbert-base-uncased-finetuned-sst-2-english"
     model = AutoModelForSequenceClassification.from_pretrained(model_name)
     tokenizer = AutoTokenizer.from_pretrained(model_name)

     tokens = tokenizer.tokenize("We are very happy to show you the Transformers library.")
     token_ids = tokenizer.convert_tokens_to_ids(tokens)
     input_ids = tokenizer("We are very happy to show you the Transformers library.")
     print(f"Takens: {tokens}")
     print("###"*10)
     print(f"Taken IDs: {token_ids}")
     print("###"*10)
     print(f"Input IDs: {input_ids}")

     X_train = ["We are very happy to show you the Transformers library.", "We hope you don't hate it."]
     batch = tokenizer(X_train, padding=True, truncation=True,
                       max_length=512, return_tensors="pt")

     print("###"*10)
     print(f"Batch: {batch}")
```

```
Takens: ['we', 'are', 'very', 'happy', 'to', 'show', 'you', 'the', 'transformers', 'library', '.']
##############################
Taken IDs: [2057, 2024, 2200, 3407, 2000, 2265, 2017, 1996, 19081, 3075, 1012]
##############################
Input IDs: {'input_ids': [101, 2057, 2024, 2200, 3407, 2000, 2265, 2017, 1996, 19081, 3075, 1012, 102], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
##############################
Batch: {'input_ids': tensor([[  101,  2057,  2024,  2200,  3407,  2000,  2265,  2017,  1996, 19081,
          3075,  1012,   102],
        [  101,  2057,  3246,  2017,  2123,  1005,  1056,  5223,  2009,  1012,
           102,     0,     0]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]])}
```

Sentiment Analysis using our model (Tokenizer) Zoom in

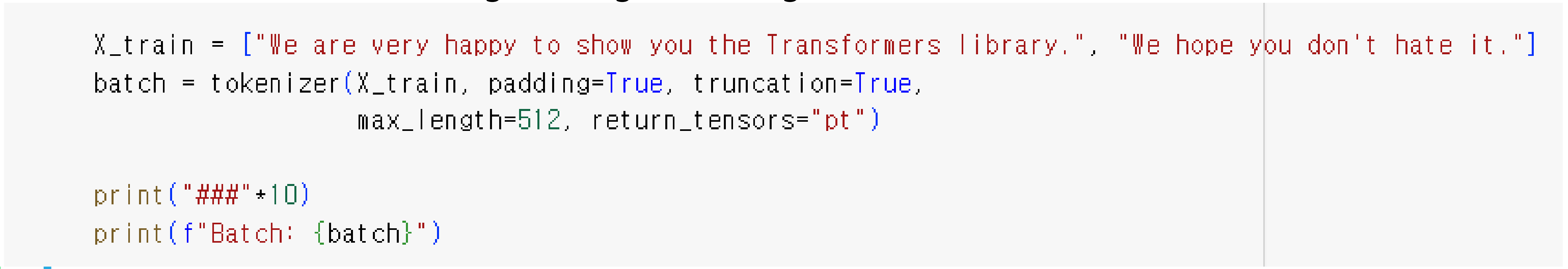
- Now we are going to use sentiment-analysis task but instead of system model we generate our model



```
tokenizer = AutoTokenizer.from_pretrained(model_name)

tokens = tokenizer.tokenize("We are very happy to show you the Transformers library.")
token_ids = tokenizer.convert_tokens_to_ids(tokens)
input_ids = tokenizer("We are very happy to show you the Transformers library.")
print(f"Takens: {tokens}")
print("###"*10)
print(f"Taken IDs: {token_ids}")
print("###"*10)
print(f"Input IDs: {input_ids}")
```

- Here define X_train which contains two sentences (different lengths)

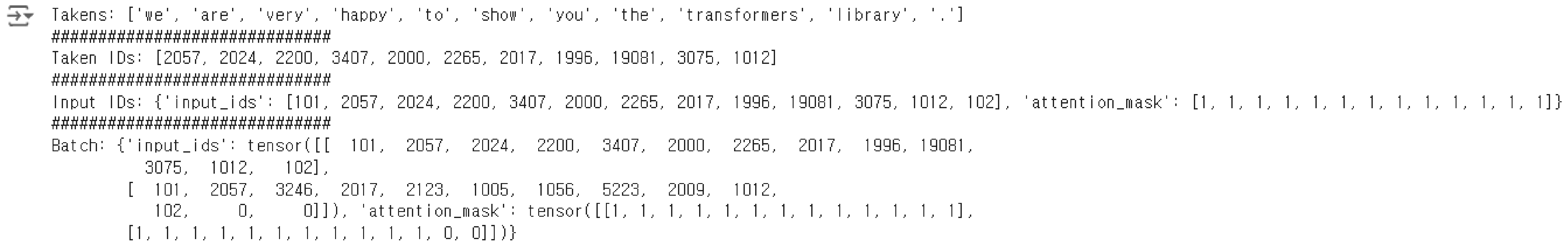
- Considers the same size during training (max_length) (512)



```
X_train = ["We are very happy to show you the Transformers library.", "We hope you don't hate it."]
batch = tokenizer(X_train, padding=True, truncation=True,
                  max_length=512, return_tensors="pt")

print("###"*10)
print(f"Batch: {batch}")
```

Sentiment Analysis using our model (Tokenizer) Zoom in

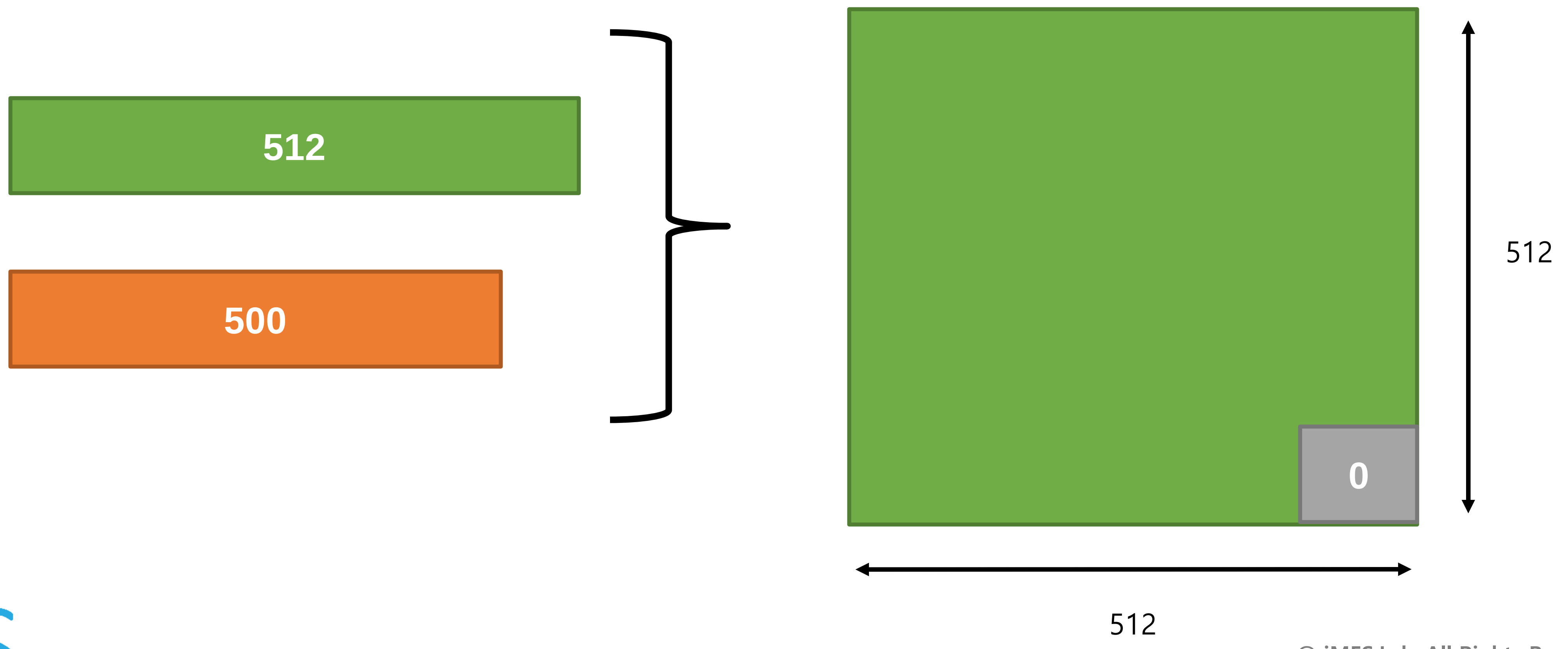
- Split a sentence to words (tokens)
- Convert each token to token ID
- Tokenizer automatically add an ID at the beginning of the dictionary
- Padding: put 0 instead of spaces
- Masking for existing words is 1, but for spaces (padding) is 0.



```
Takens: ['we', 'are', 'very', 'happy', 'to', 'show', 'you', 'the', 'transformers', 'library', '.']
################################
Taken IDs: [2057, 2024, 2200, 3407, 2000, 2265, 2017, 1996, 19081, 3075, 1012]
################################
Input IDs: {'input_ids': [101, 2057, 2024, 2200, 3407, 2000, 2265, 2017, 1996, 19081, 3075, 1012, 102], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
################################
Batch: {'input_ids': tensor([[  101,  2057,  2024,  2200,  3407,  2000,  2265,  2017,  1996, 19081,
          3075,  1012,   102],
        [  101,  2057,  3246,  2017,  2123,  1005,  1056,  5223,  2009,  1012,
           102,     0,     0]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]])}
```

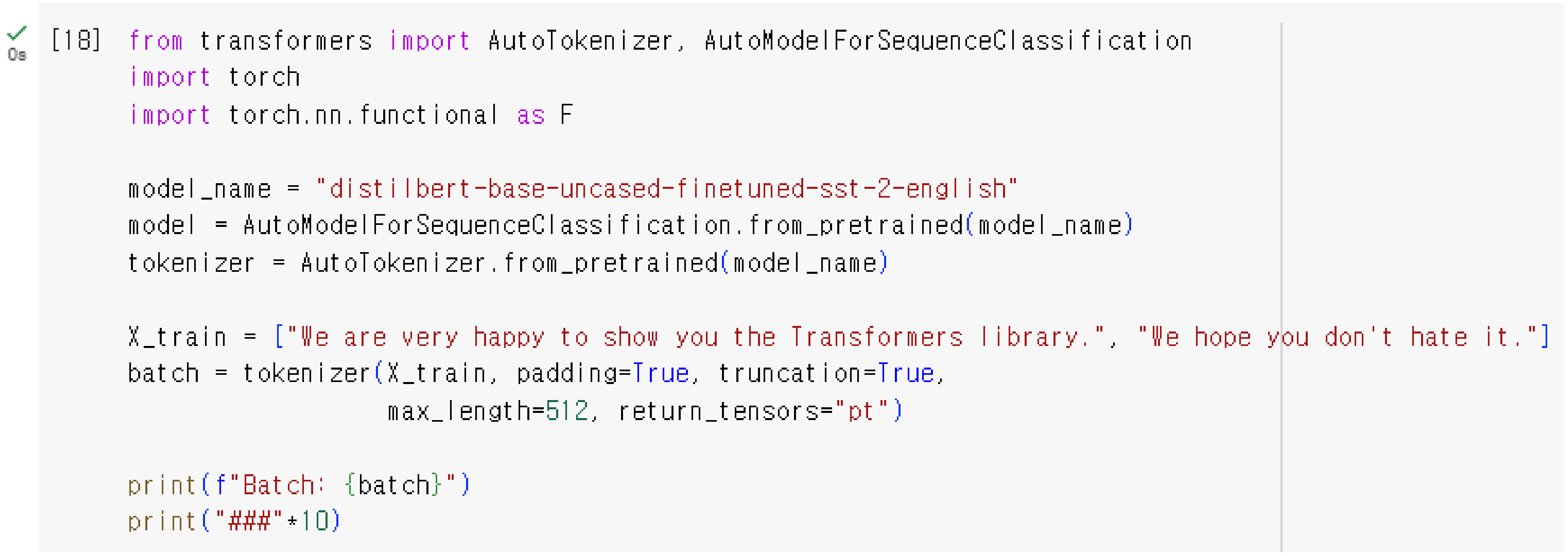
Sentiment Analysis using our model (Tokenizer) Zoom in

- During the training process, we should give tokenizer as a batch (long time training)
- What is batch?
- Batch is a matrix including the same number of rows and columns



Sentiment Analysis using our models using pytorch

- We import torch library and related modules



```
[18] from transformers import AutoTokenizer, AutoModelForSequenceClassification
     import torch
     import torch.nn.functional as F

     model_name = "distilbert-base-uncased-finetuned-sst-2-english"
     model = AutoModelForSequenceClassification.from_pretrained(model_name)
     tokenizer = AutoTokenizer.from_pretrained(model_name)

     X_train = ["We are very happy to show you the Transformers library.", "We hope you don't hate it."]
     batch = tokenizer(X_train, padding=True, truncation=True,
                       max_length=512, return_tensors="pt")

     print(f"Batch: {batch}")
     print("###"*10)
```

Sentiment Analysis using our models using pytorch

- Using softmax from F that makes the summation of weights to 1

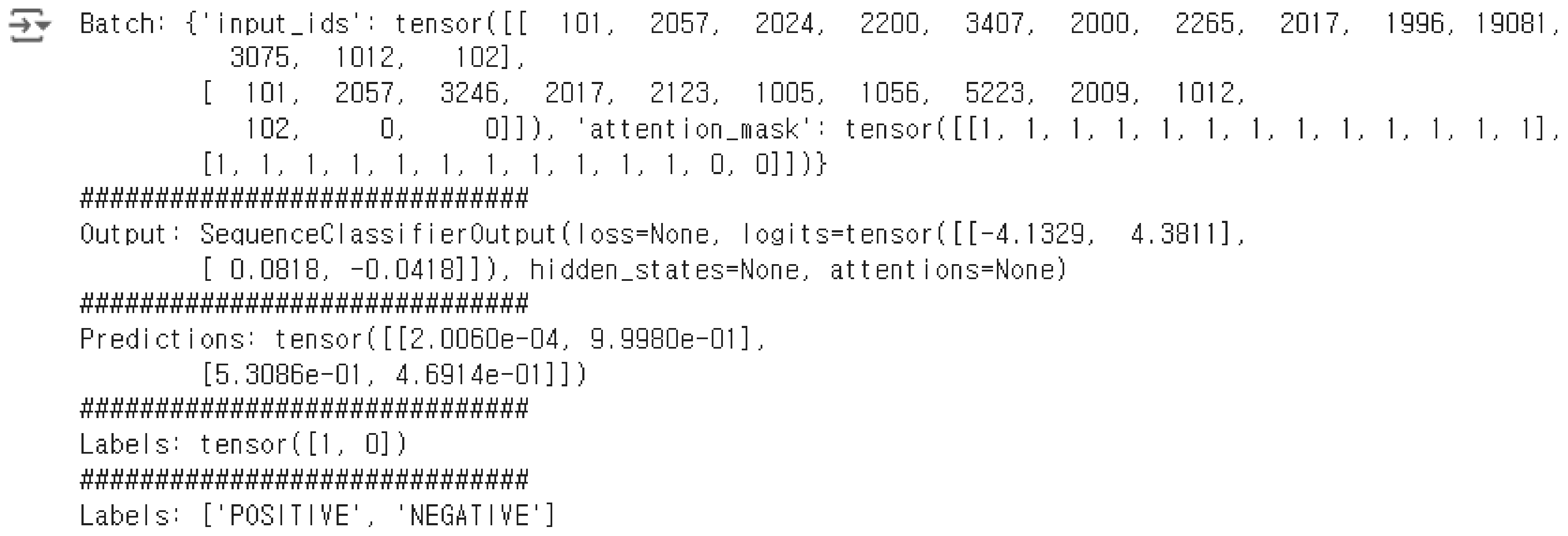
```
import torch
import torch.nn.functional as F
```

```
with torch.no_grad():
  outputs = model(**batch)
  print(f"Output: {outputs}")
  print("###"*10)
  predictions = F.softmax(outputs.logits, dim=1) # the summation of neurons' weight will be 1
  print(f"Predictions: {predictions}") # show from the highest weight in two classes
  print("###"*10)
  labels = torch.argmax(predictions, dim=1)
  print(f"Labels: {labels}")
  print("###"*10)
  labels = [model.config.id2label[label_id] for label_id in labels.tolist()]
  print(f"Labels: {labels}")
```

Put the label based on the name of class *'positive'*, *'negative'* instead of 0 or 1.

Sentiment Analysis using our models using pytorch

- We have two classes in sentiment analysis



```
Batch: {'input_ids': tensor([[  101,  2057,  2024,  2200,  3407,  2000,  2265,  2017,  1996, 19081,
          3075,  1012,   102],
        [  101,  2057,  3246,  2017,  2123,  1005,  1056,  5223,  2009,  1012,
           102,     0,     0]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]])}
##############################
Output: SequenceClassifierOutput(loss=None, logits=tensor([[-4.1329,  4.3811],
        [ 0.0818, -0.0418]]), hidden_states=None, attentions=None)
##############################
Predictions: tensor([[2.0060e-04, 9.9980e-01],
        [5.3086e-01, 4.6914e-01]])
##############################
Labels: tensor([1, 0])
##############################
Labels: ['POSITIVE', 'NEGATIVE']
```

Intelligent networking and computing for Media and Energy Systems Lab

Thank you

faranak1995@gachon.ac.kr

가천대학교 소프트웨어학과
Gachon University