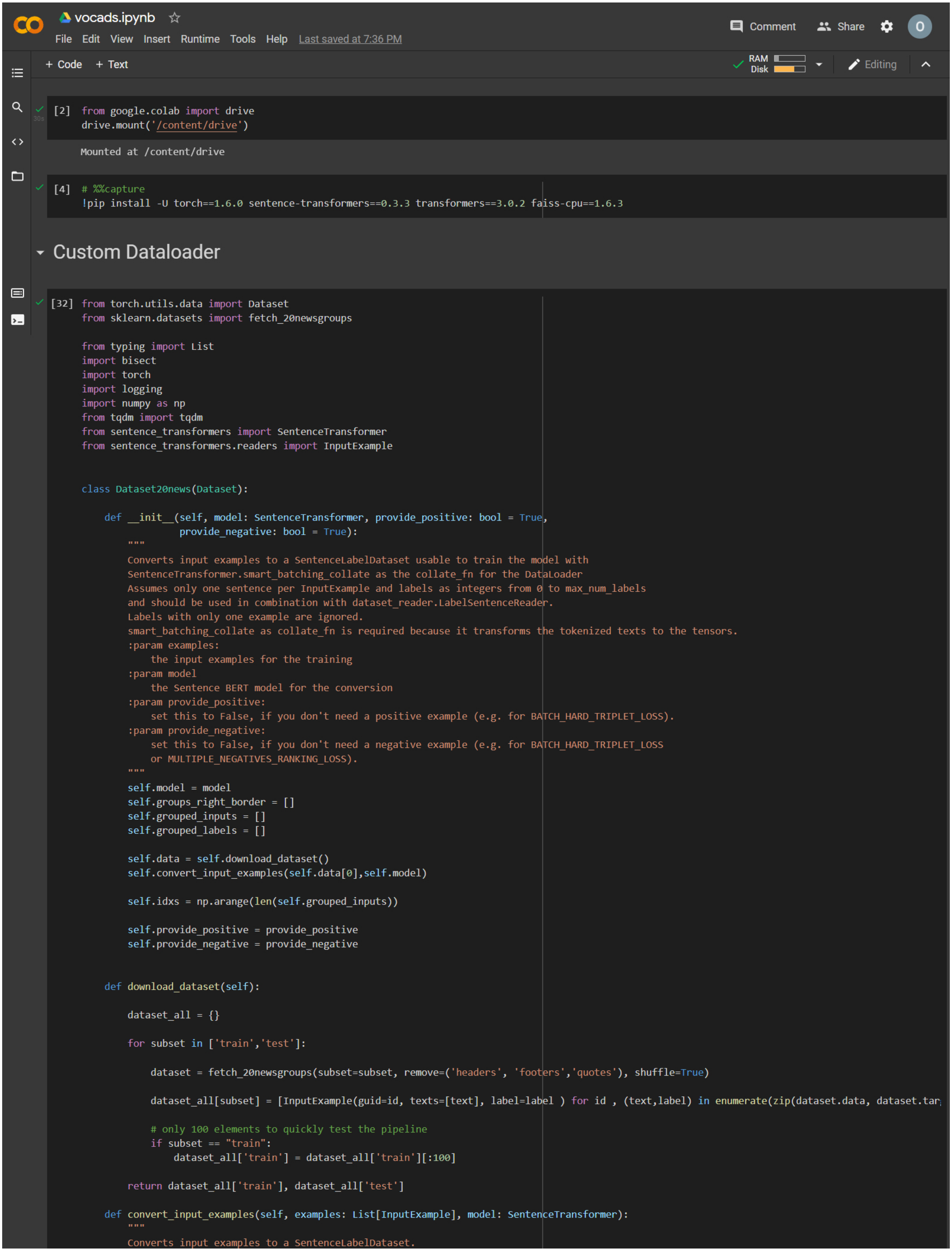


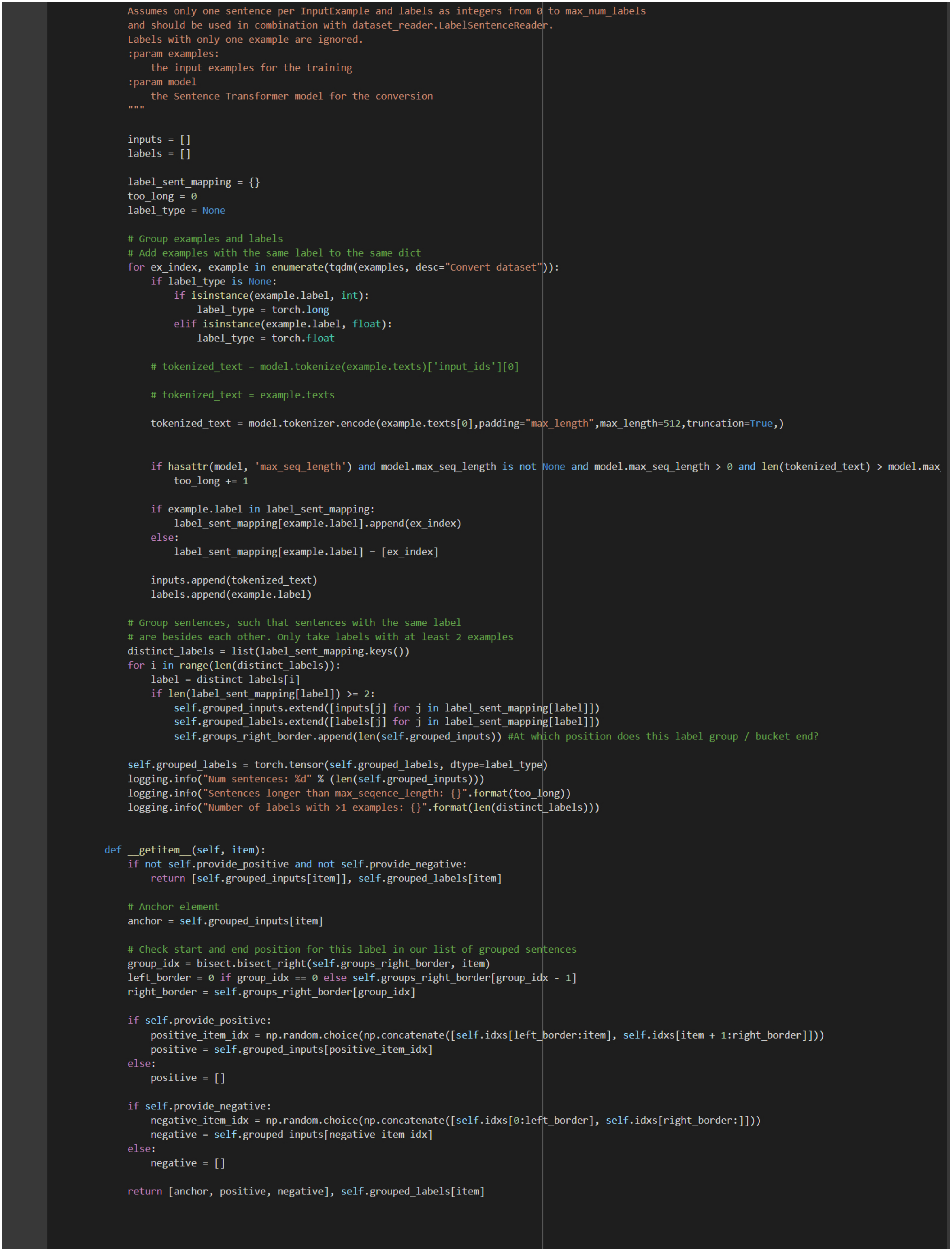
```
[2] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

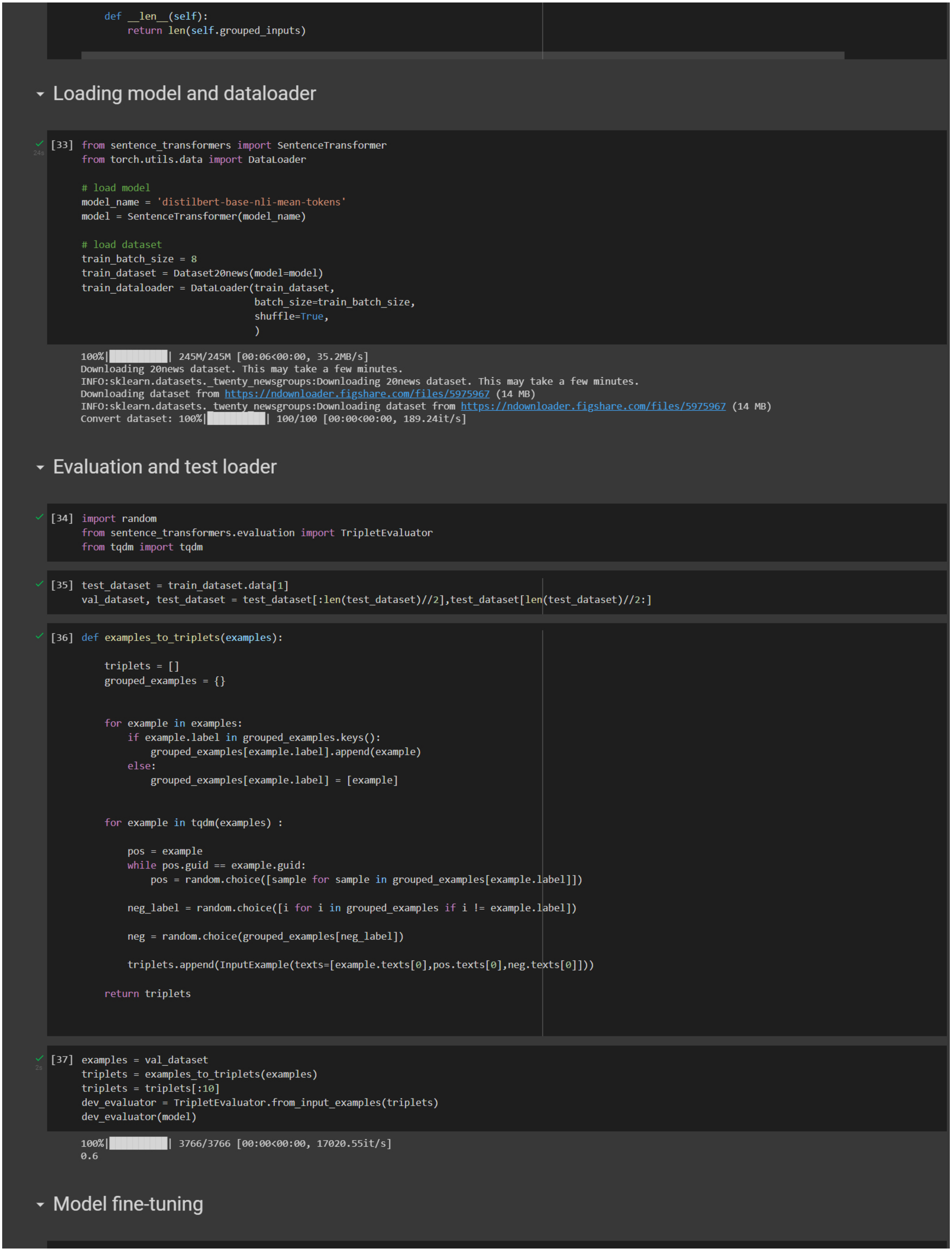
```
[4] # %%capture
    !pip install -U torch==1.6.0 sentence-transformers==0.3.3 transformers==3.0.2 faiss-cpu==1.6.3
```

Custom Dataloader

```
[32] from torch.utils.data import Dataset
     from sklearn.datasets import fetch_20newsgroups

     from typing import List
     import bisect
     import torch
     import logging
     import numpy as np
     from tqdm import tqdm
     from sentence_transformers import SentenceTransformer
     from sentence_transformers.readers import InputExample


     class Dataset20news(Dataset):

         def __init__(self, model: SentenceTransformer, provide_positive: bool = True,
                      provide_negative: bool = True):
             """
             Converts input examples to a SentenceLabelDataset usable to train the model with
             SentenceTransformer.smart_batching_collate as the collate_fn for the DataLoader
             Assumes only one sentence per InputExample and labels as integers from 0 to max_num_labels
             and should be used in combination with dataset_reader.LabelSentenceReader.
             Labels with only one example are ignored.
             smart_batching_collate as collate_fn is required because it transforms the tokenized texts to the tensors.
             :param examples:
                 the input examples for the training
             :param model
                 the Sentence BERT model for the conversion
             :param provide_positive:
                 set this to False, if you don't need a positive example (e.g. for BATCH_HARD_TRIPLET_LOSS).
             :param provide_negative:
                 set this to False, if you don't need a negative example (e.g. for BATCH_HARD_TRIPLET_LOSS
                 or MULTIPLE_NEGATIVES_RANKING_LOSS).
             """
             self.model = model
             self.groups_right_border = []
             self.grouped_inputs = []
             self.grouped_labels = []

             self.data = self.download_dataset()
             self.convert_input_examples(self.data[0],self.model)

             self.idxs = np.arange(len(self.grouped_inputs))

             self.provide_positive = provide_positive
             self.provide_negative = provide_negative


         def download_dataset(self):

             dataset_all = {}

             for subset in ['train','test']:

                 dataset = fetch_20newsgroups(subset=subset, remove=('headers', 'footers','quotes'), shuffle=True)

                 dataset_all[subset] = [InputExample(guid=id, texts=[text], label=label ) for id , (text,label) in enumerate(zip(dataset.data, dataset.tar

                 # only 100 elements to quickly test the pipeline
                 if subset == "train":
                     dataset_all['train'] = dataset_all['train'][:100]

             return dataset_all['train'], dataset_all['test']

         def convert_input_examples(self, examples: List[InputExample], model: SentenceTransformer):
             """
             Converts input examples to a SentenceLabelDataset.
```



```
        Assumes only one sentence per InputExample and labels as integers from 0 to max_num_labels
        and should be used in combination with dataset_reader.LabelSentenceReader.
        Labels with only one example are ignored.
        :param examples:
            the input examples for the training
        :param model
            the Sentence Transformer model for the conversion
        """

        inputs = []
        labels = []

        label_sent_mapping = {}
        too_long = 0
        label_type = None

        # Group examples and labels
        # Add examples with the same label to the same dict
        for ex_index, example in enumerate(tqdm(examples, desc="Convert dataset")):
            if label_type is None:
                if isinstance(example.label, int):
                    label_type = torch.long
                elif isinstance(example.label, float):
                    label_type = torch.float

            # tokenized_text = model.tokenize(example.texts)['input_ids'][0]

            # tokenized_text = example.texts

            tokenized_text = model.tokenizer.encode(example.texts[0],padding="max_length",max_length=512,truncation=True,)


            if hasattr(model, 'max_seq_length') and model.max_seq_length is not None and model.max_seq_length > 0 and len(tokenized_text) > model.max
                too_long += 1

            if example.label in label_sent_mapping:
                label_sent_mapping[example.label].append(ex_index)
            else:
                label_sent_mapping[example.label] = [ex_index]

            inputs.append(tokenized_text)
            labels.append(example.label)

        # Group sentences, such that sentences with the same label
        # are besides each other. Only take labels with at least 2 examples
        distinct_labels = list(label_sent_mapping.keys())
        for i in range(len(distinct_labels)):
            label = distinct_labels[i]
            if len(label_sent_mapping[label]) >= 2:
                self.grouped_inputs.extend([inputs[j] for j in label_sent_mapping[label]])
                self.grouped_labels.extend([labels[j] for j in label_sent_mapping[label]])
                self.groups_right_border.append(len(self.grouped_inputs)) #At which position does this label group / bucket end?

        self.grouped_labels = torch.tensor(self.grouped_labels, dtype=label_type)
        logging.info("Num sentences: %d" % (len(self.grouped_inputs)))
        logging.info("Sentences longer than max_seqence_length: {}".format(too_long))
        logging.info("Number of labels with >1 examples: {}".format(len(distinct_labels)))


    def __getitem__(self, item):
        if not self.provide_positive and not self.provide_negative:
            return [self.grouped_inputs[item]], self.grouped_labels[item]

        # Anchor element
        anchor = self.grouped_inputs[item]

        # Check start and end position for this label in our list of grouped sentences
        group_idx = bisect.bisect_right(self.groups_right_border, item)
        left_border = 0 if group_idx == 0 else self.groups_right_border[group_idx - 1]
        right_border = self.groups_right_border[group_idx]

        if self.provide_positive:
            positive_item_idx = np.random.choice(np.concatenate([self.idxs[left_border:item], self.idxs[item + 1:right_border]]))
            positive = self.grouped_inputs[positive_item_idx]
        else:
            positive = []

        if self.provide_negative:
            negative_item_idx = np.random.choice(np.concatenate([self.idxs[0:left_border], self.idxs[right_border:]]))
            negative = self.grouped_inputs[negative_item_idx]
        else:
            negative = []

        return [anchor, positive, negative], self.grouped_labels[item]
```



```
    def __len__(self):
        return len(self.grouped_inputs)
```

Loading model and dataloader

```
[33] from sentence_transformers import SentenceTransformer
     from torch.utils.data import DataLoader

     # load model
     model_name = 'distilbert-base-nli-mean-tokens'
     model = SentenceTransformer(model_name)

     # load dataset
     train_batch_size = 8
     train_dataset = Dataset20news(model=model)
     train_dataloader = DataLoader(train_dataset,
                                   batch_size=train_batch_size,
                                   shuffle=True,
                                   )
```

```
100%|██████████| 245M/245M [00:06<00:00, 35.2MB/s]
Downloading 20news dataset. This may take a few minutes.
INFO:sklearn.datasets._twenty_newsgroups:Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
INFO:sklearn.datasets._twenty_newsgroups:Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
Convert dataset: 100%|██████████| 100/100 [00:00<00:00, 189.24it/s]
```

Evaluation and test loader

```
[34] import random
     from sentence_transformers.evaluation import TripletEvaluator
     from tqdm import tqdm
```

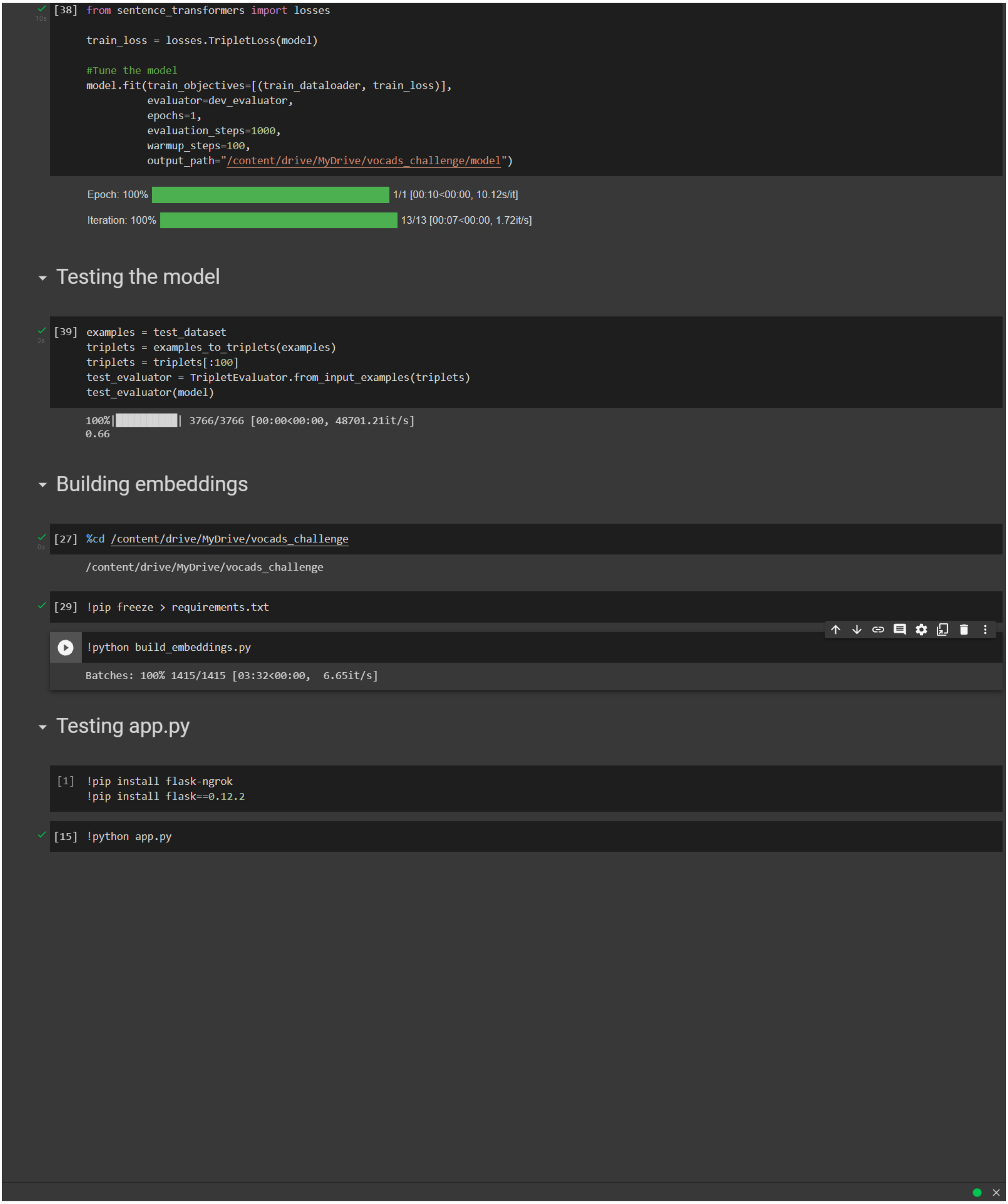
```
[35] test_dataset = train_dataset.data[1]
     val_dataset, test_dataset = test_dataset[:len(test_dataset)//2],test_dataset[len(test_dataset)//2:]
```

```
[36] def examples_to_triplets(examples):

         triplets = []
         grouped_examples = {}


         for example in examples:
             if example.label in grouped_examples.keys():
                 grouped_examples[example.label].append(example)
             else:
                 grouped_examples[example.label] = [example]


         for example in tqdm(examples) :

             pos = example
             while pos.guid == example.guid:
                 pos = random.choice([sample for sample in grouped_examples[example.label]])

             neg_label = random.choice([i for i in grouped_examples if i != example.label])

             neg = random.choice(grouped_examples[neg_label])

             triplets.append(InputExample(texts=[example.texts[0],pos.texts[0],neg.texts[0]]))

         return triplets
```

```
[37] examples = val_dataset
     triplets = examples_to_triplets(examples)
     triplets = triplets[:10]
     dev_evaluator = TripletEvaluator.from_input_examples(triplets)
     dev_evaluator(model)
```

```
100%|██████████| 3766/3766 [00:00<00:00, 17020.55it/s]
0.6
```

Model fine-tuning



```
[38] from sentence_transformers import losses

     train_loss = losses.TripletLoss(model)

     #Tune the model
     model.fit(train_objectives=[(train_dataloader, train_loss)],
               evaluator=dev_evaluator,
               epochs=1,
               evaluation_steps=1000,
               warmup_steps=100,
               output_path="/content/drive/MyDrive/vocads_challenge/model")
```

```
Epoch: 100% ██████████ 1/1 [00:10<00:00, 10.12s/it]

Iteration: 100% ██████████ 13/13 [00:07<00:00, 1.72it/s]
```

Testing the model

```
[39] examples = test_dataset
     triplets = examples_to_triplets(examples)
     triplets = triplets[:100]
     test_evaluator = TripletEvaluator.from_input_examples(triplets)
     test_evaluator(model)
```

```
100%|██████████| 3766/3766 [00:00<00:00, 48701.21it/s]
0.66
```

Building embeddings

```
[27] %cd /content/drive/MyDrive/vocads_challenge
```

```
/content/drive/MyDrive/vocads_challenge
```

```
[29] !pip freeze > requirements.txt
```

```
!python build_embeddings.py
```

```
Batches: 100% 1415/1415 [03:32<00:00,  6.65it/s]
```

Testing app.py

```
[1]  !pip install flask-ngrok
     !pip install flask==0.12.2
```

```
[15] !python app.py
```