```
!pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.17.0-py3-none-any.whl (3.8 MB)
     |████████████████████████████████| 3.8 MB 5.3 MB/s
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from transformers) (21.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (1.21.5)
Collecting huggingface-hub<1.0,>=0.1.0
  Downloading huggingface_hub-0.4.0-py3-none-any.whl (67 kB)
     |████████████████████████████████| 67 kB 5.1 MB/s
Collecting tokenizers!=0.11.3,>=0.11.1
  Downloading tokenizers-0.11.6-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (6.5 MB)
     |████████████████████████████████| 6.5 MB 29.9 MB/s
Collecting sacremoses
  Downloading sacremoses-0.0.49-py3-none-any.whl (895 kB)
     |████████████████████████████████| 895 kB 44.4 MB/s
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from transformers) (4.63.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (2019.12.20)
Collecting pyyaml>=5.1
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (596
     |████████████████████████████████| 596 kB 48.4 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers) (2.23.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from transformers) (4.11.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers) (3.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.7/dist-packages (from huggingface-hub<1.0,>
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=20.0->transf
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->transformers) (3.7
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (3.0.4
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (2021
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers) (7.1.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers) (1.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers) (1.15.0)
Installing collected packages: pyyaml, tokenizers, sacremoses, huggingface-hub, transformers
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed huggingface-hub-0.4.0 pyyaml-6.0 sacremoses-0.0.49 tokenizers-0.11.6 transformers-4.17.0
```

```
#import libraries
import os
import pandas as pd
from time import time
import numpy as np
from sklearn.metrics import accuracy_score
import torch
from torch.utils.data import Dataset, DataLoader, TensorDataset, random_split, RandomSampler, SequentialSampler
import transformers
from transformers import BertConfig, BertForSequenceClassification, RobertaTokenizerFast, AutoModelForTokenClassification, AdamW, Be
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import f1_score, accuracy_score
```

```
#download data
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
print(len(df_train), '\n', df_train.head())
print(len(df_test), '\n', df_test.head())
```

```
34647
    id                                       comment_text  toxicity
0   0                 fuck you you self righteous creep          3
1   1   stop stop the goddam vandalism or there ll be...         2
2   2  i agree rt does have a few shortcomings  but i...         0
3   3  if you would like verfiability here is the lin...         0
4   4  do you think there s consensus for me to be on...         0
9194
       id                                       comment_text
0  34647  oh that great repository of free cultural work...
1  34648  my rfa  with apologies for the impersonal awb ...
2  34649  it looks like a number of articles you created...
3  34650  oh  but i see you ve been  block  for other  s...
4  34651  accord of the discussion in mariah carey compo...
```

Preparing dataloader

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
```

```
Downloading: 100%                                                    226k/226k [00:00<00:00, 1.05MB/s]
```

```
df_train.dropna(inplace=True)
df_test.dropna(inplace=True)
```

```
df_train['comment_text'] = df_train['comment_text'].apply(lambda x: str(x).strip())
train_sentences = list(df_train.comment_text)
train_labels = list(df_train.toxicity)
df_test['comment_text'] = df_test['comment_text'].apply(lambda x: str(x).strip())
test_sentences = list(df_test.comment_text)
```

```
def get_max_len(sentences: list):
  max = 0
  sentence0 = ''
  for sentence in sentences:
    max = len(sentence) if len(sentence) > max else max
    sentence0 = sentence if len(sentence) >= max else sentence0
  return max, sentence0

MAX_LENGTH = 300
CLASSES_NUMBER = len(set(train_labels))
```

```
def tokenize(sentence: list, tokenizer, max_length):
    """
    input: one sentence ['some', 'sentence', 'here'];
    output: tokens_ids - tensor([1234, 23, 3241]),
            attention_mask -  tensor([1, 1, 1, 1, 1, 1, 1, 1, 0,]),
            bert_tokens - ['what', 'both', 'al', 'pac', '##ino', 'and', 'robert', 'den'].
    """
    tokens_ids = []
    attention_mask = []
    bert_tokens = []
    tokenized_input = tokenizer(sentence,
                                add_special_tokens = True,
                                truncation = True,
                                max_length = max_length,
                                pad_to_max_length = True,
                                return_attention_mask = True,
                                return_tensors = 'pt',
                                is_split_into_words=False
                               )

    tokens_ids = tokenized_input['input_ids'][0]
    attention_mask = tokenized_input['attention_mask'][0]
    bert_tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"][0])
    return tokens_ids, attention_mask, bert_tokens
```

```
#tokenize
# lists of bert tokens
bert_tokens = []
#tokens ids
tokens_ids = []
# attention masks
attentions_masks = []
# tokenize each sentence
for sentence in train_sentences:
    token_id, attention_mask, bert_token = tokenize(sentence, tokenizer, MAX_LENGTH)
    tokens_ids.append(token_id)
    attentions_masks.append(attention_mask)
    bert_tokens.append(bert_token)
```

```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2277: FutureWarning: The `pad_to_max_length` arg
  FutureWarning,
```

```
#for test sentences
#tokenize
# lists of bert tokens
test_bert_tokens = []
#tokens ids
test_tokens_ids = []
# attention masks
test_attentions_masks = []
# tokenize each sentence
for sentence in test_sentences:
    token_id, attention_mask, bert_token = tokenize(sentence, tokenizer, MAX_LENGTH)
    test_tokens_ids.append(token_id)
    test_attentions_masks.append(attention_mask)
    test_bert_tokens.append(bert_token)
```

```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2277: FutureWarning: The `pad_to_max_length` arg
  FutureWarning,
```

```
len(bert_tokens)
```

```
34646
```

```
len(train_labels)
```

```
34646
```

```
print("Bert tokens: ", bert_tokens[0])
print("Tokens ids: ", tokens_ids[0])
print("New labels: ", train_labels[0])
print("Attention mask: ", attentions_masks[0])
```

```
Bert tokens:  ['[CLS]', 'fuck', 'you', 'you', 'self', 'righteous', 'creep', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD
Tokens ids:  tensor([  101,  6616,  2017,  2017,  2969, 19556, 19815,   102,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0])
New labels:  3
Attention mask:  tensor([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Convert input_ids, attentions_masks and train_labels to TensorDataset

```
pt_input_ids = torch.stack(tokens_ids, dim=0)

pt_attention_masks = torch.stack(attentions_masks, dim=0)

pt_labels = torch.tensor(train_labels, dtype=torch.long)
```

```
#for test sentences
test_pt_input_ids = torch.stack(test_tokens_ids, dim=0)

test_pt_attention_masks = torch.stack(test_attentions_masks, dim=0)
```

```
# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(pt_input_ids, pt_attention_masks, pt_labels)

# Create a 90-10 train-validation split.
train_size = int(0.999 * len(dataset))
```

```
val_size = len(dataset) - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

print('{:>5,} training samples'.format(train_size))
print('{:>5,} validation samples'.format(val_size))
```

```
34,611 training samples
   35 validation samples
```

```
#for test dataset
test_dataset = TensorDataset(test_pt_input_ids, test_pt_attention_masks)
# Divide the dataset by randomly selecting samples.

print('{:>5,} test samples'.format(len(test_dataset)))
```

```
9,194 test samples
```

Convert TensorDataset to Dataloader

```
BATCH_SIZE = 16

train_dataloader = DataLoader(train_dataset, sampler=RandomSampler(train_dataset), batch_size=BATCH_SIZE)
validation_dataloader = DataLoader(val_dataset, sampler=SequentialSampler(val_dataset), batch_size=BATCH_SIZE)
```

```
test_dataloader = DataLoader(test_dataset, sampler=SequentialSampler(test_dataset), batch_size=BATCH_SIZE)
```

```
# Load BertForSequenceClassification, the pretrained BERT model with a single linear classification layer on top.

model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased', # Use the 124-layer, 1024-hidden, 16-heads, 340M parameters BERT model with an uncased vocab.
    num_labels = CLASSES_NUMBER, # The number of output labels--2 for binary classification. You can increase this for multi-class t
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForSequenceClassification: ['cls
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another tas
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
# If there's a GPU available...

if torch.cuda.is_available():

    # Tell PyTorch to use the GPU.

    device = torch.device('cuda')


    print('There are %d GPU(s) available.' % torch.cuda.device_count())

    print('We will use the GPU:', torch.cuda.get_device_name(0))

# If not...

else:
    print('No GPU available, using the CPU instead.')
    device = torch.device('cpu')

model.to(device)
```

```
            )
            (10): BertLayer(
              (attention): BertAttention(
                (self): BertSelfAttention(
                  (query): Linear(in_features=768, out_features=768, bias=True)
                  (key): Linear(in_features=768, out_features=768, bias=True)
                  (value): Linear(in_features=768, out_features=768, bias=True)
                  (dropout): Dropout(p=0.1, inplace=False)
                )
                (output): BertSelfOutput(
                  (dense): Linear(in_features=768, out_features=768, bias=True)
                  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                  (dropout): Dropout(p=0.1, inplace=False)
                )
              )
              (intermediate): BertIntermediate(
```

```
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (11): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=6, bias=True)
)
```

Training

```
# Load the AdamW optimizer
optimizer = AdamW(model.parameters(),
                  lr = 5e-5, # args.learning_rate
                  eps = 1e-8 # args.adam_epsilon
                )
```

```
/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:309: FutureWarning: This implementation of AdamW is deprec
  FutureWarning,
```

```
# Number of training epochs
epochs = 2

# Total number of training steps is number of batches * number of epochs.
total_steps = len(train_dataloader) * epochs

# Create the learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0,
                                            num_training_steps = total_steps)
```

```
import random

seed_val = 42

random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

loss_values = []

for epoch_i in range(0, epochs):

    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    total_loss = 0

    model.train()
```

```
    for step, batch in enumerate(train_dataloader):

        if step % 100 == 0 and not step == 0:

            # Report progress.
            print('  Batch {:>5,}  of  {:>5,}.'.format(step, len(train_dataloader)))

        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        model.zero_grad()

        outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)

        loss = outputs[0]

        total_loss += loss.item()

        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        optimizer.step()

        scheduler.step()

    avg_train_loss = total_loss / len(train_dataloader)
    loss_values.append(avg_train_loss)

    print("  Average training loss: {0:.2f}".format(avg_train_loss))
```

```
 ======== Epoch 1 / 2 ========
 Training...
   Batch   100  of  2,164.
   Batch   200  of  2,164.
   Batch   300  of  2,164.
   Batch   400  of  2,164.
   Batch   500  of  2,164.
   Batch   600  of  2,164.
   Batch   700  of  2,164.
   Batch   800  of  2,164.
   Batch   900  of  2,164.
   Batch 1,000  of  2,164.
   Batch 1,100  of  2,164.
   Batch 1,200  of  2,164.
   Batch 1,300  of  2,164.
   Batch 1,400  of  2,164.
   Batch 1,500  of  2,164.
   Batch 1,600  of  2,164.
   Batch 1,700  of  2,164.
   Batch 1,800  of  2,164.
   Batch 1,900  of  2,164.
   Batch 2,000  of  2,164.
   Batch 2,100  of  2,164.
   Average training loss: 0.77
 ======== Epoch 2 / 2 ========
 Training...
   Batch   100  of  2,164.
   Batch   200  of  2,164.
   Batch   300  of  2,164.
   Batch   400  of  2,164.
   Batch   500  of  2,164.
   Batch   600  of  2,164.
   Batch   700  of  2,164.
   Batch   800  of  2,164.
   Batch   900  of  2,164.
   Batch 1,000  of  2,164.
   Batch 1,100  of  2,164.
   Batch 1,200  of  2,164.
   Batch 1,300  of  2,164.
   Batch 1,400  of  2,164.
   Batch 1,500  of  2,164.
   Batch 1,600  of  2,164.
   Batch 1,700  of  2,164.
   Batch 1,800  of  2,164.
   Batch 1,900  of  2,164.
   Batch 2,000  of  2,164.
   Batch 2,100  of  2,164.
   Average training loss: 0.56
```

```
# Put model in evaluation mode
model.eval()

# Tracking variables
predictions, true_labels = [], []
```

```
# Predict
for batch in validation_dataloader:
    # Add batch to GPU
    batch = tuple(t.to(device) for t in batch) ##t.to(device)

    # Unpack the inputs from our dataloader
    b_input_ids, b_input_mask, b_labels = batch

    # Telling the model not to compute or store gradients, saving memory and

    with torch.no_grad():
        # Forward pass, calculate logit predictions
        outputs = model(b_input_ids, token_type_ids=None,
                        attention_mask=b_input_mask)

    logits = outputs[0]

    # Move logits and labels to CPU
    logits = logits.detach().cpu().numpy()
    label_ids = b_labels.to('cpu').numpy()

    # Store predictions and true labels
    predictions.append(logits)
    true_labels.append(label_ids)

print('    DONE.')
```

```
    DONE.
```

```
#Test dataset
# Put model in evaluation mode
model.eval()

# Tracking variables
predictions_test = []

# Predict
for batch in test_dataloader:
    # Add batch to GPU
    batch = tuple(t.to(device) for t in batch) ##t.to(device)

    # Unpack the inputs from our dataloader
    b_input_ids, b_input_mask = batch

    # Telling the model not to compute or store gradients, saving memory and

    with torch.no_grad():
        # Forward pass, calculate logit predictions
        outputs = model(b_input_ids, token_type_ids=None,
                        attention_mask=b_input_mask)

    logits = outputs[0]

    # Move logits and labels to CPU
    logits = logits.detach().cpu().numpy()
    #label_ids = b_labels.to('cpu').numpy()

    # Store predictions and true labels
    predictions_test.append(logits)
    #true_labels.append(label_ids)

print('    DONE.')
```

```
    DONE.
```

```
submission = pd.read_csv('submission.csv')
submission.head()
```

	id	prediction
0	34647	0
1	34648	0
2	34649	0
3	34650	0
4	34651	0

```
predictions_test_0 = np.concatenate(predictions_test, axis=0)
predictions_test_0
```

```
array([[ 5.7892346 ,  2.9139578 , -0.7423075 , -2.3402362 , -3.460374  ,
        -2.8209934 ],
       [ 7.370486  ,  0.5536067 , -1.7443788 , -2.4937348 , -2.5184624 ,
        -1.96266   ],
       [ 7.213716  , -0.06232698, -1.8447533 , -2.3940964 , -2.1407135 ,
        -1.3780146 ],
       ...,
       [ 1.3675389 ,  3.7210956 ,  1.6388266 , -0.774359  , -2.7443225 ,
        -4.286558  ],
       [-2.3002644 ,  0.6463383 ,  0.68021655,  1.6560824 ,  0.9153693 ,
        -1.596381  ],
       [-0.12701559,  2.6128974 ,  2.931504  ,  0.5130284 , -2.853128  ,
        -4.496347  ]], dtype=float32)
```

```
all_predictions_test = np.argmax(predictions_test_0, axis=1)
len(all_predictions_test)
```

```
9194
```

```
all_predictions_test[-1]
```

```
2
```

```
submission['prediction'] = all_predictions_test.tolist()
```

```
submission
```

	id	prediction
0	34647	0
1	34648	0
2	34649	0
3	34650	2
4	34651	1
...	...	...
9189	43836	4
9190	43837	2
9191	43838	1
9192	43839	3
9193	43840	2

9194 rows × 2 columns

```
all_predictions = np.concatenate(predictions, axis=0)
all_true_labels = np.concatenate(true_labels, axis=0)
```

```
predict = np.argmax(all_predictions, axis=1)
```

```
true = all_true_labels
```

```
from sklearn.metrics import f1_score

f1 = f1_score(predict, true, average='micro')

print ("F1 score: {:.2%}".format(f1))
```

```
F1 score: 74.29%
```

```
submission.to_csv('submission_test.csv', index=False)
```

```
ff = pd.read_csv('submission_test.csv')
```

```
ff
```

	id	prediction
0	34647	0
1	34648	0
2	34649	0
3	34650	2
4	34651	1
...	...	...
9189	43836	4
9190	43837	2
9193	43840	2

9194 rows × 2 columns