In [53]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

from sklearn.metrics import accuracy_score, classification_report, confusion_mat
rix

import torch
from torchtext.data import Field, TabularDataset, BucketIterator, Iterator

from transformers import RobertaTokenizer, RobertaModel, AdamW, get_linear_sched
ule_with_warmup

import warnings
warnings.filterwarnings('ignore')

import logging
logging.getLogger("transformers.tokenization_utils_base").setLevel(logging.ERROR
)
```

In [54]:

```
df = pd.read_csv("../input/ieeefnid/fakenn.csv")
print(df.shape)
df.head()
```

```
(17326, 8)
```

Out[54]:

	id	date	speaker	statement	sources
0	1636	2010-03-28T17:45:34-04:00	Charlie Crist	Rubio's tax swap proposal "would have been a m...	['http://blogs.tampabay.com/buzz/files/0403
1	4352	2011-08-29T06:00:00-04:00	Bobby Scott	"The estimated savings of this (debt ceiling) ...	['http://www.bobbyscott.house.gov/index.php
2	16471	2019-02-12T17:35:38-05:00	Wisconsin Republican Legislative leaders	Foxconn has already "made a positive impact ac...	['https://www.wispolitics.com/2019/sen-fitzg
3	1557	2010-03-05T18:24:02-05:00	Dave Aronberg	Says Gov. Charlie Crist has called him "a rock...	['http://www.davearonberg.com/about', 'http:
4	12826	2016-07-29T18:09:31-04:00	Jeannette Vaught	"Only five Texas counties account for almost 9...	['http://www.mystatesman.com/news/news/c

In [55]:

```
#df.isna()
indexNames = df[ df['label_fnn'] == "label_fnn" ].index
# Delete these row indexes from dataFrame
df.drop(indexNames , inplace=True)
print(df.shape)
```

```
(17324, 8)
```

In [56]:

```
encode_label = {'fake' : 0, 'real' : 1}
df['label'] = df['label_fnn'].map(encode_label)
df = df.astype({"label": int})
df['all'] = df['statement'] + ". " + df['fullText_based_content']
df['DATE'] = pd.to_datetime(df['date'],utc=True, errors='coerce')
df['MONTH'] = df['DATE'].dt.month
df['year'] = df['DATE'].dt.year
df = df.drop(["date"], axis =1)
df.head()
```

Out[56]:

	id	speaker	statement	sources	paragr
0	1636	Charlie Crist	Rubio's tax swap proposal "would have been a m...	['http://blogs.tampabay.com/buzz/files/040307l...	['Gov. launch to ...
1	4352	Bobby Scott	"The estimated savings of this (debt ceiling) ...	['http://www.bobbyscott.house.gov/index.php?op...	['U.S. I D-3rd,
2	16471	Wisconsin Republican Legislative leaders	Foxconn has already "made a positive impact ac...	['https://www.wispolitics.com/2019/sen-fitzger...	["Amid questi Techn
3	1557	Dave Aronberg	Says Gov. Charlie Crist has called him "a rock...	['http://www.davearonberg.com/about', 'http://...	["State Aronb candi..
4	12826	Jeannette Vaught	"Only five Texas counties account for almost 9...	['http://www.mystatesman.com/news/news/opinion...	['From Rio Gr

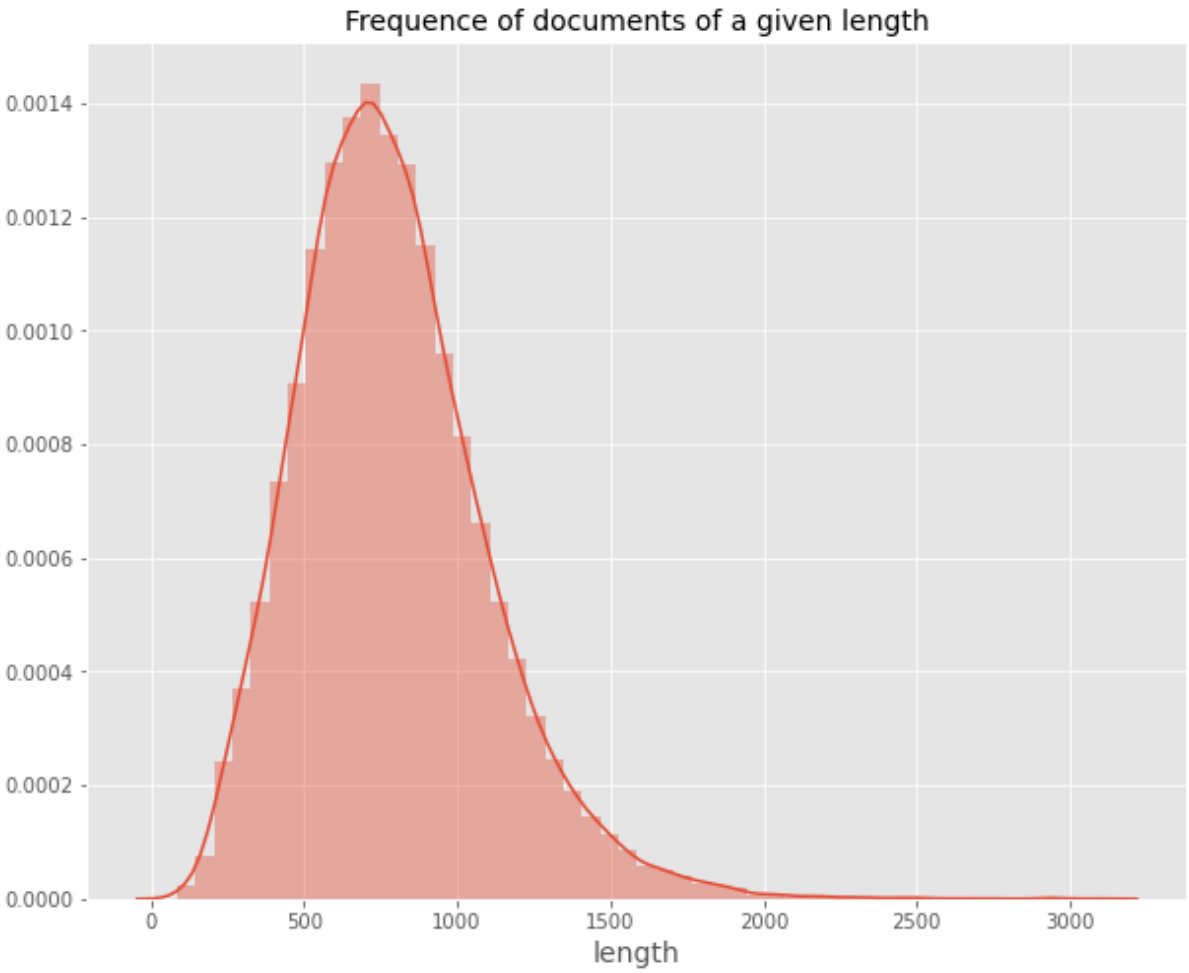
In [57]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17324 entries, 0 to 17325
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      17324 non-null  object
 1   speaker                 17324 non-null  object
 2   statement               17324 non-null  object
 3   sources                 17324 non-null  object
 4   paragraph_based_content  17324 non-null  object
 5   fullText_based_content  17324 non-null  object
 6   label_fnn               17324 non-null  object
 7   label                   17324 non-null  int64
 8   all                     17324 non-null  object
 9   DATE                    17324 non-null  datetime64[ns, UTC]
 10  MONTH                   17324 non-null  int64
 11  year                    17324 non-null  int64
dtypes: datetime64[ns, UTC](1), int64(3), object(8)
memory usage: 1.7+ MB
```

In [58]:

```
# Plot histogram with the length. Truncate max length to 5000 tokens.
plt.style.use("ggplot")

plt.figure(figsize=(10, 8))
df['length'] = df['all'].apply(lambda x: len(x.split()))
sns.distplot(df[df['length'] < 5000]['length'])
plt.title('Frequence of documents of a given length', fontsize=14)
plt.xlabel('length', fontsize=14)
None
```



In [59]:

```
# Set random seed and set device to GPU.
torch.manual_seed(17)

if torch.cuda.is_available():
    device = torch.device('cuda:0')
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
else:
    device = torch.device('cpu')

print(device)
```

```
cuda:0
```

In [60]:

```
# Initialize tokenizer.
tokenizer = RobertaTokenizer.from_pretrained("roberta-base")
```

In [61]:

```
df.to_csv("data_new.csv")
```

In [62]:

```
# Set tokenizer hyperparameters.
MAX_SEQ_LEN = 256
BATCH_SIZE = 16
PAD_INDEX = tokenizer.convert_tokens_to_ids(tokenizer.pad_token)
UNK_INDEX = tokenizer.convert_tokens_to_ids(tokenizer.unk_token)


# Define columns to read.
label_field = Field(sequential=False, use_vocab=False, batch_first=True)
text_field = Field(use_vocab=False,
                   tokenize=tokenizer.encode,
                   include_lengths=False,
                   batch_first=True,
                   fix_length=MAX_SEQ_LEN,
                   pad_token=PAD_INDEX,
                   unk_token=UNK_INDEX)

fields = {'all' : ('all', text_field), 'label' : ('label', label_field)}


# Read preprocessed CSV into TabularDataset and split it into train, test and vali
d.
train_data, valid_data, test_data = TabularDataset(path=f"./data_new.csv",
                                                   format='CSV',
                                                   fields=fields,
                                                   skip_header=False).split(spli
t_ratio=[0.70, 0.2, 0.1],
                                                                            stra
tified=True,
                                                                            stra
ta_field='label')

# Create train and validation iterators.
train_iter, valid_iter = BucketIterator.splits((train_data, valid_data),
                                               batch_size=BATCH_SIZE,
                                               device=device,
                                               shuffle=True,
                                               sort_key=lambda x: len(x.all),
                                               sort=True,
                                               sort_within_batch=False)

# Test iterator, no shuffling or sorting required.
```

```
test_iter = Iterator(test_data, batch_size=BATCH_SIZE, device=device, train=Fals
e, shuffle=False, sort=False)
```

In [63]:

```
# Functions for saving and loading model parameters and metrics.
def save_checkpoint(path, model, valid_loss):
    torch.save({'model_state_dict': model.state_dict(),
                'valid_loss': valid_loss}, path)


def load_checkpoint(path, model):
    state_dict = torch.load(path, map_location=device)
    model.load_state_dict(state_dict['model_state_dict'])

    return state_dict['valid_loss']


def save_metrics(path, train_loss_list, valid_loss_list, global_steps_list):
    state_dict = {'train_loss_list': train_loss_list,
                'valid_loss_list': valid_loss_list,
                'global_steps_list': global_steps_list}

    torch.save(state_dict, path)


def load_metrics(path):
    state_dict = torch.load(path, map_location=device)
    return state_dict['train_loss_list'], state_dict['valid_loss_list'], state_d
ict['global_steps_list']
```

In [64]:

```
# Model with extra layers on top of RoBERTa
class ROBERTAClassifier(torch.nn.Module):
    def __init__(self, dropout_rate=0.3):
        super(ROBERTAClassifier, self).__init__()

        self.roberta = RobertaModel.from_pretrained('roberta-base')
        self.d1 = torch.nn.Dropout(dropout_rate)
        self.l1 = torch.nn.Linear(768, 64)
        self.bn1 = torch.nn.LayerNorm(64)
        self.d2 = torch.nn.Dropout(dropout_rate)
        self.l2 = torch.nn.Linear(64, 2)

    def forward(self, input_ids, attention_mask):
        _, x = self.roberta(input_ids=input_ids, attention_mask=attention_mask)
        x = self.d1(x)
        x = self.l1(x)
        x = self.bn1(x)
        x = torch.nn.Tanh()(x)
        x = self.d2(x)
        x = self.l2(x)

        return x
```

In [65]:

```
def pretrain(model,
             optimizer,
             train_iter,
             valid_iter,
             scheduler = None,
             valid_period = len(train_iter),
             num_epochs = 5):

    # Pretrain linear layers, do not train bert
    for param in model.roberta.parameters():
        param.requires_grad = False

    model.train()

    # Initialize losses and loss histories
    train_loss = 0.0
    valid_loss = 0.0
    global_step = 0

    # Train loop
    for epoch in range(num_epochs):
        for (source, target), _ in train_iter:
            mask = (source != PAD_INDEX).type(torch.uint8)

            y_pred = model(input_ids=source,
                           attention_mask=mask)

            loss = torch.nn.CrossEntropyLoss()(y_pred, target)

            loss.backward()

            # Optimizer and scheduler step
            optimizer.step()
            scheduler.step()

            optimizer.zero_grad()

            # Update train loss and global step
            train_loss += loss.item()
            global_step += 1

            # Validation loop. Save progress and evaluate model performance.
            if global_step % valid_period == 0:
```

```
                model.eval()

                with torch.no_grad():
                    for (source, target), _ in valid_iter:
                        mask = (source != PAD_INDEX).type(torch.uint8)
                        y_pred = model(input_ids=source,
                                       attention_mask=mask)

                        loss = torch.nn.CrossEntropyLoss()(y_pred, target)

                        valid_loss += loss.item()

                # Store train and validation loss history
                train_loss = train_loss / valid_period
                valid_loss = valid_loss / len(valid_iter)

                model.train()

                # print summary
                print('Epoch [{}/{}], global step [{}/{}], PT Loss: {:.4f}, Val
 Loss: {:.4f}'
                      .format(epoch+1, num_epochs, global_step, num_epochs*len(t
rain_iter),
                              train_loss, valid_loss))

                train_loss = 0.0
                valid_loss = 0.0

    # Set bert parameters back to trainable
    for param in model.roberta.parameters():
        param.requires_grad = True

    print('Pre-training done!')
```

In [66]:

```
def train(model,
          optimizer,
          train_iter,
          valid_iter,
          scheduler = None,
          num_epochs = 5,
          valid_period = len(train_iter)):

    # Initialize losses and loss histories
    train_loss = 0.0
    valid_loss = 0.0
    train_loss_list = []
    valid_loss_list = []
    best_valid_loss = float('Inf')

    global_step = 0
    global_steps_list = []

    model.train()

    # Train loop
    for epoch in range(num_epochs):
        for (source, target), _ in train_iter:
            mask = (source != PAD_INDEX).type(torch.uint8)

            y_pred = model(input_ids=source,
                           attention_mask=mask)
            #output = model(input_ids=source,
            #               labels=target,
            #               attention_mask=mask)

            loss = torch.nn.CrossEntropyLoss()(y_pred, target)
            #loss = output[0]

            loss.backward()

            #torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)

            # Optimizer and scheduler step
            optimizer.step()
            scheduler.step()

            optimizer.zero_grad()
```

```
            # Update train loss and global step
            train_loss += loss.item()
            global_step += 1

            # Validation loop. Save progress and evaluate model performance.
            if global_step % valid_period == 0:
                model.eval()

                with torch.no_grad():
                    for (source, target), _ in valid_iter:
                        mask = (source != PAD_INDEX).type(torch.uint8)

                        y_pred = model(input_ids=source,
                                       attention_mask=mask)
                        #output = model(input_ids=source,
                        #               labels=target,
                        #               attention_mask=mask)

                        loss = torch.nn.CrossEntropyLoss()(y_pred, target)
                        #loss = output[0]

                        valid_loss += loss.item()

                # Store train and validation loss history
                train_loss = train_loss / valid_period
                valid_loss = valid_loss / len(valid_iter)
                train_loss_list.append(train_loss)
                valid_loss_list.append(valid_loss)
                global_steps_list.append(global_step)

                # print summary
                print('Epoch [{}/{}], global step [{}/{}], Train Loss: {:.4f}, V
alid Loss: {:.4f}'
                      .format(epoch+1, num_epochs, global_step, num_epochs*len(t
rain_iter),
                              train_loss, valid_loss))

                # checkpoint
                if best_valid_loss > valid_loss:
                    best_valid_loss = valid_loss
                    save_checkpoint('/model.pkl', model, best_valid_loss)
                    save_metrics('/metric.pkl', train_loss_list, valid_loss_list
, global_steps_list)

                train_loss = 0.0
```

```
                valid_loss = 0.0
                model.train()

    save_metrics('/metric.pkl', train_loss_list, valid_loss_list, global_steps_l
ist)
    print('Training done!')
```

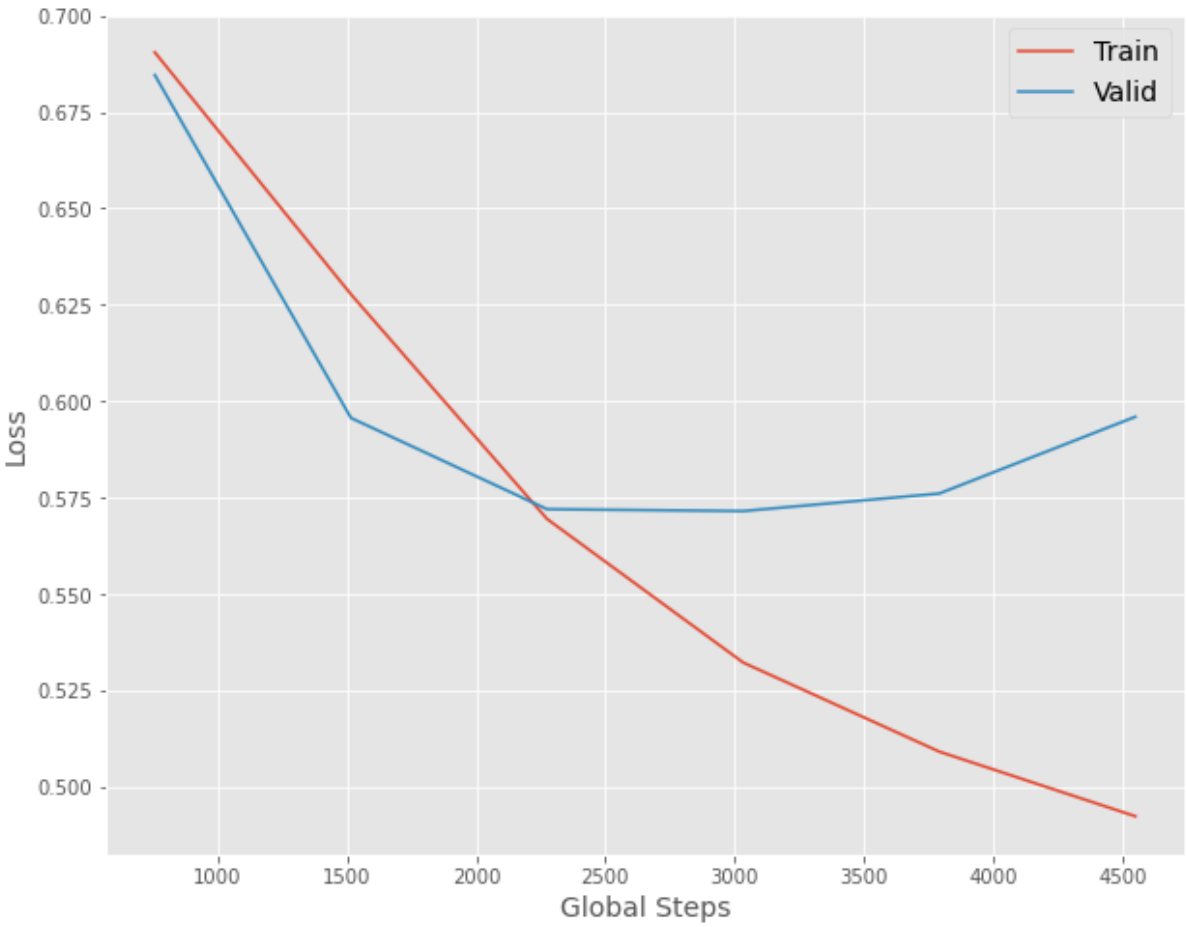
In [67]:

```
NUM_EPOCHS = 3
steps_per_epoch = len(train_iter)

model = ROBERTAClassifier(0.4)
model = model.to(device)


optimizer = AdamW(model.parameters(), lr=1e-4)
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=steps_per_epoch*1,
                                            num_training_steps=steps_per_epoch*N
UM_EPOCHS)

print("======================== Start pretraining ==============================="
)

pretrain(model=model,
         train_iter=train_iter,
         valid_iter=valid_iter,
         optimizer=optimizer,
         scheduler=scheduler,
         num_epochs=NUM_EPOCHS)

NUM_EPOCHS = 6
print("======================== Start training ==================================="
)
optimizer = AdamW(model.parameters(), lr=2e-6)
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=steps_per_epoch*2,
                                            num_training_steps=steps_per_epoch*N
UM_EPOCHS)

train(model=model,
      train_iter=train_iter,
      valid_iter=valid_iter,
      optimizer=optimizer,
      scheduler=scheduler,
      num_epochs=NUM_EPOCHS)
```

```
====================== Start pretraining ==========================
====
Epoch [1/3], global step [758/2274], PT Loss: 0.7102, Val Loss: 0.69
66
Epoch [2/3], global step [1516/2274], PT Loss: 0.6939, Val Loss: 0.6
962
Epoch [3/3], global step [2274/2274], PT Loss: 0.6903, Val Loss: 0.6
896
Pre-training done!
====================== Start training ============================
====
Epoch [1/6], global step [758/4548], Train Loss: 0.6905, Valid Loss:
0.6846
Epoch [2/6], global step [1516/4548], Train Loss: 0.6278, Valid Los
s: 0.5957
Epoch [3/6], global step [2274/4548], Train Loss: 0.5695, Valid Los
s: 0.5720
Epoch [4/6], global step [3032/4548], Train Loss: 0.5323, Valid Los
s: 0.5715
Epoch [5/6], global step [3790/4548], Train Loss: 0.5092, Valid Los
s: 0.5761
Epoch [6/6], global step [4548/4548], Train Loss: 0.4924, Valid Los
s: 0.5960
Training done!
```

In [68]:

```
plt.figure(figsize=(10, 8))
train_loss_list, valid_loss_list, global_steps_list = load_metrics('/metric.pkl'
)
plt.plot(global_steps_list, train_loss_list, label='Train')
plt.plot(global_steps_list, valid_loss_list, label='Valid')
plt.xlabel('Global Steps', fontsize=14)
plt.ylabel('Loss', fontsize=14)
plt.legend(fontsize=14)
plt.show()
```



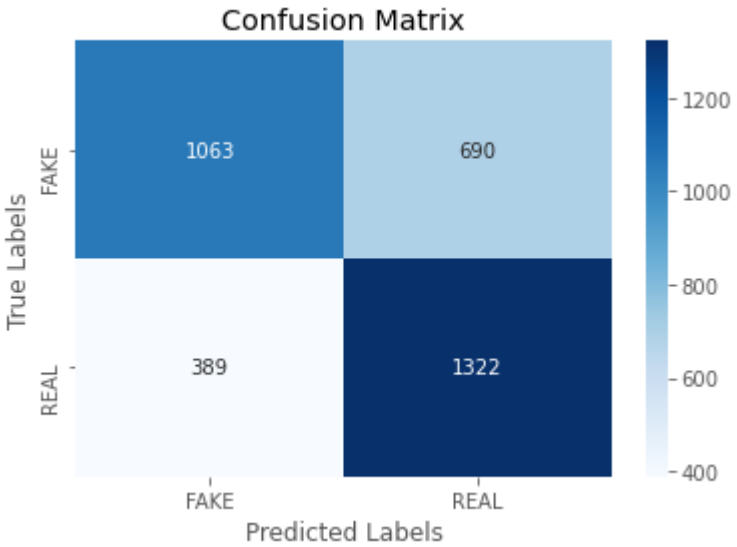
In [69]:

```
def evaluate(model, test_loader):
    y_pred = []
    y_true = []

    model.eval()
    with torch.no_grad():
        for (source, target), _ in test_loader:
                mask = (source != PAD_INDEX).type(torch.uint8)

                output = model(source, attention_mask=mask)

                y_pred.extend(torch.argmax(output, axis=-1).tolist())
                y_true.extend(target.tolist())

    print('Classification Report:')
    print(classification_report(y_true, y_pred, labels=[1,0], digits=4))

    cm = confusion_matrix(y_true, y_pred, labels=[1,0])
    ax = plt.subplot()

    sns.heatmap(cm, annot=True, ax = ax, cmap='Blues', fmt="d")

    ax.set_title('Confusion Matrix')

    ax.set_xlabel('Predicted Labels')
    ax.set_ylabel('True Labels')

    ax.xaxis.set_ticklabels(['FAKE', 'REAL'])
    ax.yaxis.set_ticklabels(['FAKE', 'REAL'])
```

In [70]:

```
model = ROBERTAClassifier()
model = model.to(device)

load_checkpoint('/model.pkl', model)

evaluate(model, test_iter)
```

```
Classification Report:
              precision    recall  f1-score   support

           1     0.7321    0.6064    0.6633      1753
           0     0.6571    0.7726    0.7102      1711

    accuracy                         0.6885      3464
   macro avg     0.6946    0.6895    0.6868      3464
weighted avg     0.6950    0.6885    0.6865      3464
```



In [71]:

```
print(len(train_data))
print(len(valid_data))
print(len(test_data))
```

```
12127
1733
3464
```