Twitter: NER NLP

Problem Statement

Twitter is a microblogging and social networking service on which users post and interact with messages known as "tweets". Every second, on average, around 6,000 tweets are tweeted on Twitter, corresponding to over 350,000 tweets sent per minute, 500 million tweets per day.

Twitter wants to automatically tag and analyze tweets for better understanding of the trends and topics without being dependent on the hashtags that the users use. Many users do not use hashtags or sometimes use wrong or mis-spelled tags, so they want to completely remove this problem and create a system of recognizing important content of the tweets.

Named Entity Recognition (NER) is an important subtask of information extraction that seeks to locate and recognise named entities.

You need to train models that will be able to identify the various named entities.

Data Description

Dataset is annotated with 10 fine-grained NER categories: person, geo-location, company, facility, product,music artist, movie, sports team, tv show and other. Dataset was extracted from tweets and is structured in CoNLL format., in English language. Containing in Text file format.

The CoNLL format is a text file with one word per line with sentences separated by an empty line. The first word in a line should be the word and the last word should be the label.

1. Import the Data and Understand the Structure of Data :

In [2]:

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

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder
import plotly.express as px
```

In [3]:

```
def load_data(filename: str):
  #conll file is stored as{token,tag} pairs, one per line
  #extracting data from conll files
  with open(filename, 'r') as file:
    lines=[line[:-1].split() for line in file] #skipping last line as it will be bla
  samples,start=[],0
  for end, parts in enumerate(lines):
    if not parts:
      sample=[(token,tag) for token,tag in lines[start:end]]
      samples.append(sample)
      start=end+1
  if start < end:
    samples.append(lines[start:end])
  return samples
```

In [4]:

```
train_samples = load_data('wnut 16.txt.conll')
test_samples = load_data('wnut 16test.txt.conll')
samples = train_samples + test_samples
schema = ['_'] + sorted({tag for sentence in samples
                             for _,tag in sentence}) # '_' is used to indicate a nul
```

In [5]:

```
len(train_samples)
```

Out[5]:

```
2394
```

In [6]:

```
len(test_samples)
```

Out[6]:

```
3850
```

In [7]:

```
schema
```

Out[7]:

```
['_',
 'B-company',
 'B-facility',
 'B-geo-loc',
 'B-movie',
 'B-musicartist',
 'B-other',
 'B-person',
 'B-product',
 'B-sportsteam',
 'B-tvshow',
 'I-company',
 'I-facility',
 'I-geo-loc',
 'I-movie',
 'I-musicartist',
 'I-other',
 'I-person',
 'I-product',
 'I-sportsteam',
 'I-tvshow',
 'O']
```

In [8]:

```
train_samples[1]
```

Out[8]:

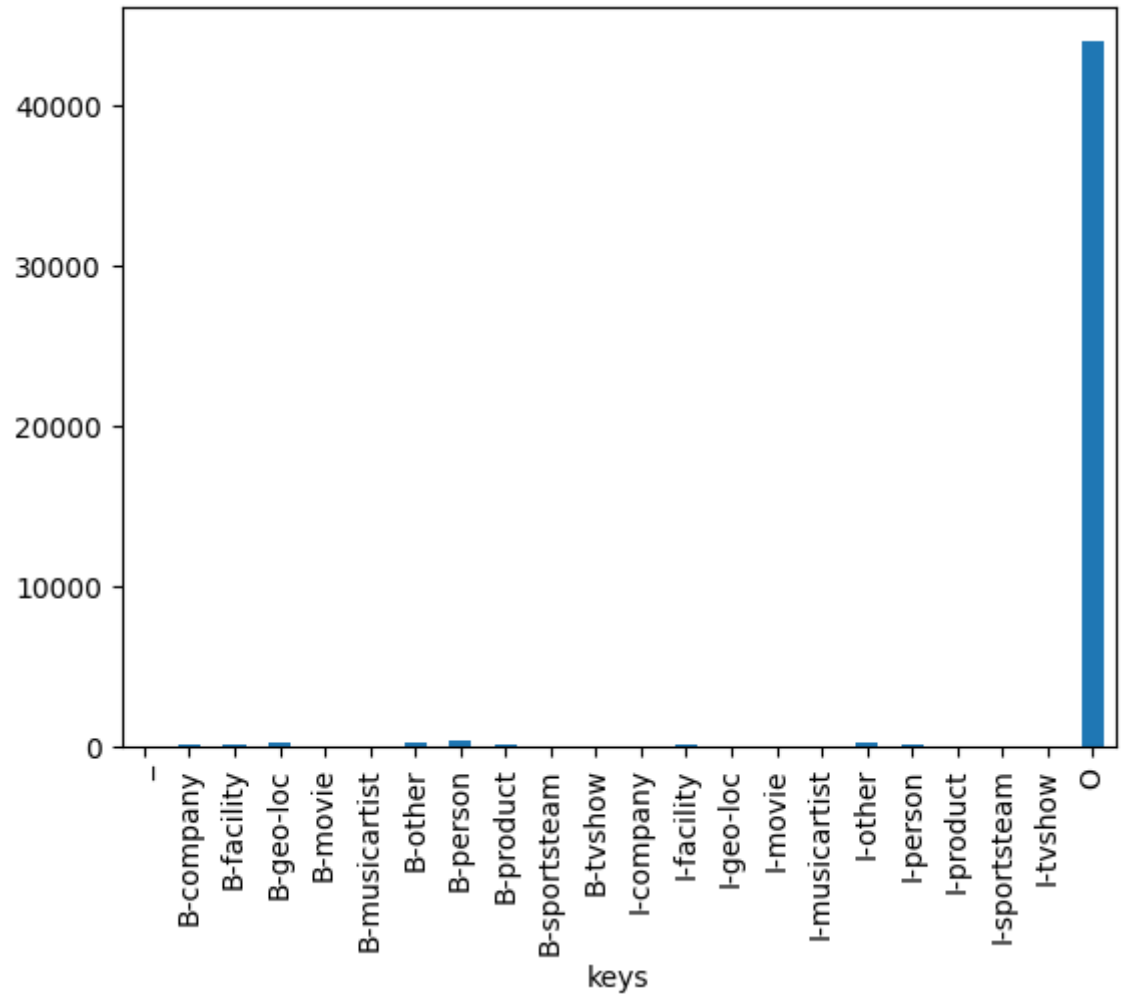
```
[('Made', 'O'),
 ('it', 'O'),
 ('back', 'O'),
 ('home', 'O'),
 ('to', 'O'),
 ('GA', 'B-geo-loc'),
 ('.', 'O'),
 ('It', 'O'),
 ('sucks', 'O'),
 ('not', 'O'),
 ('to', 'O'),
 ('be', 'O'),
 ('at', 'O'),
 ('Disney', 'B-facility'),
 ('world', 'I-facility'),
 (',', 'O'),
 ('but', 'O'),
 ('its', 'O'),
 ('good', 'O'),
 ('to', 'O'),
 ('be', 'O'),
 ('home', 'O'),
 ('.', 'O'),
 ('Time', 'O'),
 ('to', 'O'),
 ('start', 'O'),
 ('planning', 'O'),
 ('the', 'O'),
 ('next', 'O'),
 ('Disney', 'B-facility'),
 ('World', 'I-facility'),
 ('trip', 'O'),
 ('.', 'O')]
```

In [9]:

```
counts={}

for tag in schema:
  counts[tag]=0
  for sample in train_samples:
    for label in sample:
      if label[1] == tag:
        counts[tag]+=1

counts_df=pd.DataFrame({'keys':list(counts.keys()),'values':list(counts.values())})
counts_df.plot.bar(x='keys',y='values',legend=False)
```

Out[9]:

```
<Axes: xlabel='keys'>
```

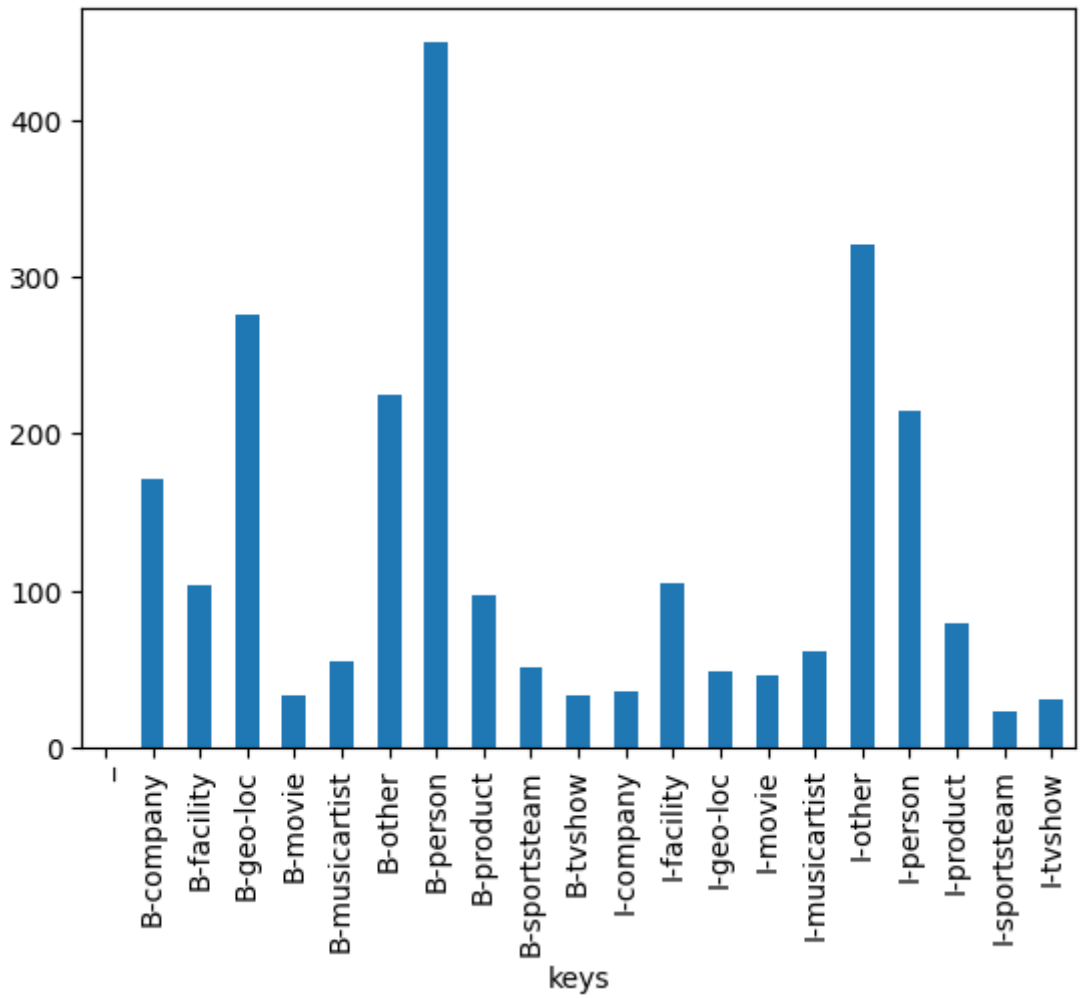


In [10]:

```
counts.pop('O')
counts_df=pd.DataFrame({'keys':list(counts.keys()),'values':list(counts.values())})
counts_df.plot.bar(x='keys',y='values',legend=False)
```

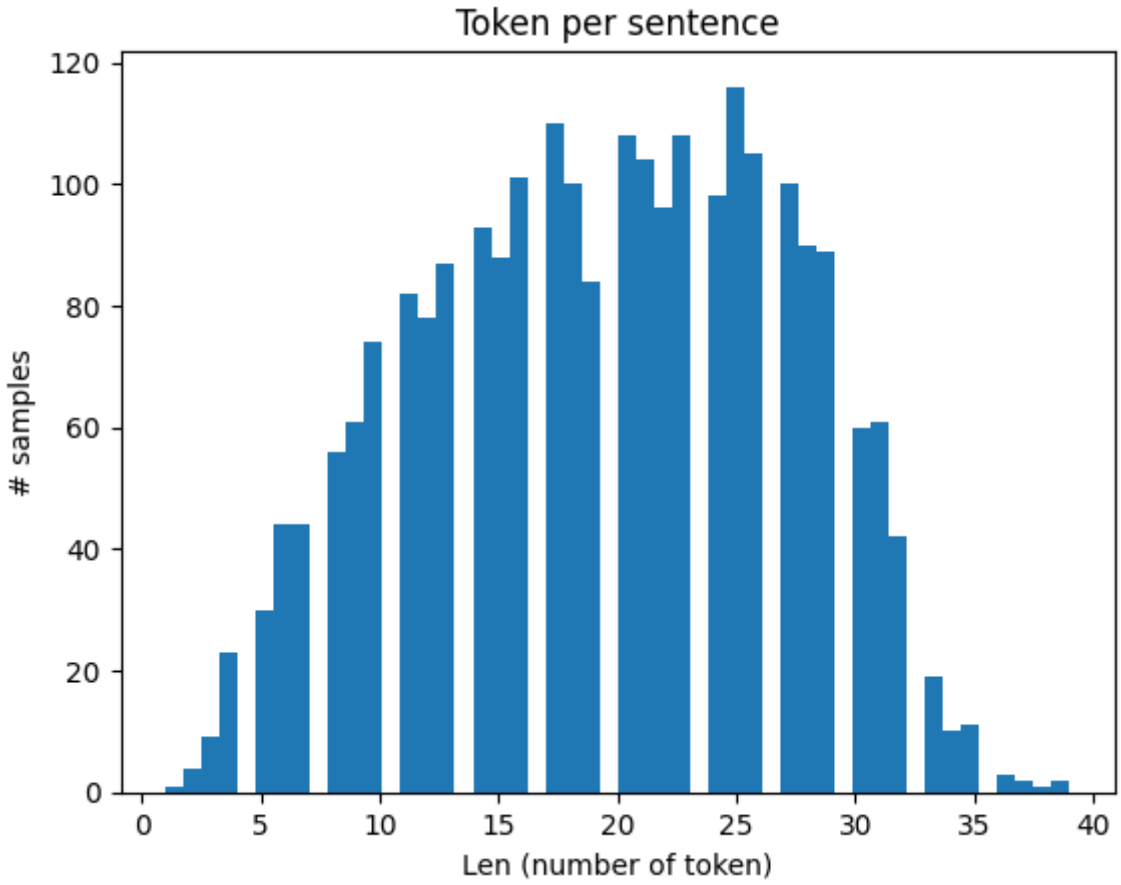
Out[10]:

```
<Axes: xlabel='keys'>
```



In [11]:

```
# Get all the sentences
# Plot sentence by lenght
plt.hist([len(s) for s in train_samples], bins=50)
plt.title('Token per sentence')
plt.xlabel('Len (number of token)')
plt.ylabel('# samples')
plt.show()
```



In [12]:

```
def create_sentence_df(data, sentence_number):
    words, tags = zip(*data)
    df = pd.DataFrame({'word': words, 'tag': tags})
    df['sentence'] = sentence_number
    return df

sentence_df_list_train = []
sentence_df_list_test = []
for idx, sentence_data in enumerate(train_samples, 1):
    sentence_df_list_train.append(create_sentence_df(sentence_data, idx))

for idx, sentence_data in enumerate(test_samples, 1):
    sentence_df_list_test.append(create_sentence_df(sentence_data, idx))

df_train = pd.concat(sentence_df_list_train, ignore_index=True)
df_test = pd.concat(sentence_df_list_test, ignore_index=True)
desired_order = ['sentence', 'word', 'tag']

# Reindex the DataFrame with the desired column order
df_train = df_train.reindex(columns=desired_order)
df_test = df_test.reindex(columns=desired_order)

print('df_train data:')
print(df_train.head())
print('-----------------------------------\n')
print('df_test data:')
print(df_test.head())
```

```
df_train data:
   sentence             word tag
0         1  @SammieLynnsMom   O
1         1         @tg10781   O
2         1             they   O
3         1             will   O
4         1               be   O
-----------------------------------

df_test data:
   sentence     word      tag
0         1      New  B-other
1         1  Orleans  I-other
2         1   Mother  I-other
3         1       's  I-other
4         1      Day  I-other
```

In [13]:

```
df_train.shape
```

Out[13]:

```
(46468, 3)
```

In [14]:

```
print("Number of sentences: ", len(df_train.groupby(['sentence'])))
```

```
Number of sentences:  2394
```

In [15]:

```
words = list(set(df_train["word"].values))
n_words = len(words)
print("Number of unique words in the dataset: ", n_words)
```

```
Number of unique words in the dataset:  10586
```

In [16]:

```
n_tags = len(df_train.groupby(['tag']))
print("Number of Labels: ", n_tags)
```

```
Number of Labels:  21
```

In [17]:

```
tags = list(set(df_train["tag"].values))
print("Tags:", tags)
```

```
Tags: ['I-tvshow', 'I-other', 'B-other', 'I-sportsteam', 'I-company',
'I-facility', 'B-geo-loc', 'O', 'I-product', 'I-person', 'B-tvshow',
'B-musicartist', 'I-movie', 'I-geo-loc', 'B-company', 'B-product', 'B-
facility', 'I-musicartist', 'B-person', 'B-movie', 'B-sportsteam']
```

In [18]:

```
#let's check the frequency of each tag
df_train['tag'].value_counts()
```

Out[18]:

```
O               44006
B-person          449
I-other           320
B-geo-loc         276
B-other           225
I-person          215
B-company         171
I-facility        105
B-facility        104
B-product          97
I-product          80
I-musicartist      61
B-musicartist      55
B-sportsteam       51
I-geo-loc          49
I-movie            46
I-company          36
B-movie            34
B-tvshow           34
I-tvshow           31
I-sportsteam       23
Name: tag, dtype: int64
```

In [19]:

```
import sklearn

# Vocabulary Key:word -> Value:token_index
# The first 2 entries are reserved for PAD and UNK
word2idx = {w: i + 2 for i, w in enumerate(words)}
word2idx["UNK"] = 1 # Unknown words
word2idx["PAD"] = 0 # Padding

# Vocabulary Key:token_index -> Value:word
idx2word = {i: w for w, i in word2idx.items()}

# Vocabulary Key:Label/Tag -> Value:tag_index
# The first entry is reserved for PAD
tag2idx = {t: i+1 for i, t in enumerate(tags)}
tag2idx["PAD"] = 0

# Vocabulary Key:tag_index -> Value:Label/Tag
idx2tag = {i: w for w, i in tag2idx.items()}
print("The word Disney is identified by the index: {}".format(word2idx["Disney"]))
print("The labels B-person(beginning of a named entity) is identified by the index:
```

```
The word Disney is identified by the index: 2351
The labels B-person(beginning of a named entity) is identified by the
index: 19
```

In [20]:

```
!pip install keras-preprocessing
```

```
Requirement already satisfied: keras-preprocessing in /usr/local/lib/p
ython3.10/dist-packages (1.1.2)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.
10/dist-packages (from keras-preprocessing) (1.22.4)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.1
0/dist-packages (from keras-preprocessing) (1.16.0)
```

In [21]:

```
MAX_LEN=39 #max length of number of tokens present in train_samples

#from keras.preprocessing.sequence import pad_sequences
from keras_preprocessing.sequence import pad_sequences

# Convert each sentence from list of Token to list of word_index
X = [[word2idx[w[0]] for w in s] for s in train_samples]

# Padding each sentence to have the same lenght
X = pad_sequences(maxlen=MAX_LEN, sequences=X, padding="post", value=word2idx["PAD"]

# Convert Tag/Label to tag_index
y = [[tag2idx[w[1]] for w in s] for s in train_samples]

# Padding each sentence to have the same lenght
y = pad_sequences(maxlen=MAX_LEN, sequences=y, padding="post", value=tag2idx["PAD"])
from keras.utils import to_categorical

# One-Hot encode
y = [to_categorical(i, num_classes=n_tags+1) for i in y]  # n_tags+1(PAD)

#Train-Test split
from sklearn.model_selection import train_test_split
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.1)
print(X_tr.shape, X_te.shape, np.array(y_tr).shape, np.array(y_te).shape)
```

```
(2154, 39) (240, 39) (2154, 39, 22) (240, 39, 22)
```

In [22]:

```
print('Raw Sample:\n ', ' '.join([w[0] for w in train_samples[1]]))
print('\n ' )
print('Raw Label:\n ', ' '.join([w[1] for w in train_samples[1]]))
print('\n ' )
print('After processing, sample:\n', X[0])
print('\n ' )
print('After processing, labels:\n', y[0])
```

```
Raw Sample:
  Made it back home to GA . It sucks not to be at Disney world , but i
ts good to be home . Time to start planning the next Disney World trip
.


Raw Label:
  O O O O O B-geo-loc O O O O O O O B-facility I-facility O O O O O O
O O O O O O O O B-facility I-facility O O


After processing, sample:
 [1474 8344  575 5856 7683 3978 6068 6648 9473 9381 5506 2855    0
0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0]


After processing, labels:
 [[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

In [23]:

```
y_tr = np.array(y_tr)
y_te = np.array(y_te)
```

In [24]:

```
!pip install tensorflow_addons
```

```
Requirement already satisfied: tensorflow_addons in /usr/local/lib/pyt
hon3.10/dist-packages (0.21.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/
dist-packages (from tensorflow_addons) (23.1)
Requirement already satisfied: typeguard<3.0.0,>=2.7 in /usr/local/li
b/python3.10/dist-packages (from tensorflow_addons) (2.13.3)
```

In [24]:

```

```

In [25]:

```
from keras.models import Model
import tensorflow as tf
from tensorflow.keras.layers import Input
from tensorflow_addons.utils.types import FloatTensorLike, TensorLike

# LSTM components
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirecti

# CRF layer
from tensorflow_addons.layers import CRF

# Sigmoid focal cross entropy loss. works well with highly unbalanced input data
from tensorflow_addons.losses import SigmoidFocalCrossEntropy
from tensorflow_addons.optimizers import AdamW


def build_model(max_len = 39, input_dim = 10590, embedding_dim = 200):
  # Model definition
  input = Input(shape=(max_len,))

  # Get embeddings
  embeddings = Embedding(input_dim=input_dim,
                       output_dim=embedding_dim,
                       input_length=max_len, mask_zero=True,
                      )(input)

  # variational biLSTM
  output_sequences = Bidirectional(LSTM(units=50, return_sequences=True))(embeddings

  # Stacking
  output_sequences = Bidirectional(LSTM(units=50, return_sequences=True))(output_seq

  # Adding more non-linearity
  dense_out = TimeDistributed(Dense(25, activation="relu"))(output_sequences)

  # CRF layer
  crf = CRF(22, name='crf') # y_tr.shape=(2154, 39, 22) so using 22 in CRF
  predicted_sequence, potentials, sequence_length, crf_kernel = crf(dense_out)

  model = Model(input, potentials)
  model.compile(
      optimizer=AdamW(weight_decay=0.001),
      metrics='accuracy',
      loss= SigmoidFocalCrossEntropy()) # Sigmoid focal cross entropy loss

  return model

model = build_model()

# Checkpointing
save_model = tf.keras.callbacks.ModelCheckpoint(filepath='twitter_ner_crf.h5',
  monitor='val_loss',
  save_weights_only=True,
  save_best_only=True,
  verbose=1
)

# Early stopping
es = tf.keras.callbacks.EarlyStopping(monitor='val_loss', verbose=1, patience=10)
```

```
callbacks = [save_model, es]

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 39)]              0

 embedding (Embedding)       (None, 39, 200)           2118000

 bidirectional (Bidirectiona  (None, 39, 100)          100400
 l)

 bidirectional_1 (Bidirectio  (None, 39, 100)          60400
 nal)

 time_distributed (TimeDistr  (None, 39, 25)           2525
 ibuted)

 crf (CRF)                   [(None, 39),              1100
                              (None, 39, 22),
                              (None,),
                              (22, 22)]

=================================================================
Total params: 2,282,425
Trainable params: 2,282,425
Non-trainable params: 0
_________________________________________________________________
```

In [26]:

```
import datetime

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_fre
history = model.fit(X_tr, y_tr, validation_data = (X_te, y_te),epochs = 10, shuffle
```

```
Epoch 1/10

WARNING:tensorflow:Gradients do not exist for variables ['chain_kerne
l:0'] when minimizing the loss. If you're using `model.compile()`, did
you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['chain_kerne
l:0'] when minimizing the loss. If you're using `model.compile()`, did
you forget to provide a `loss` argument?

68/68 [==============================] - 69s 400ms/step - loss: 0.5934
- accuracy: 0.9271 - val_loss: 0.3993 - val_accuracy: 0.9760
Epoch 2/10
68/68 [==============================] - 12s 174ms/step - loss: 0.3505
- accuracy: 0.9734 - val_loss: 0.3130 - val_accuracy: 0.9760
Epoch 3/10
68/68 [==============================] - 12s 178ms/step - loss: 0.2767
- accuracy: 0.9734 - val_loss: 0.2540 - val_accuracy: 0.9760
Epoch 4/10
68/68 [==============================] - 13s 191ms/step - loss: 0.2232
- accuracy: 0.9734 - val_loss: 0.2117 - val_accuracy: 0.9760
Epoch 5/10
68/68 [==============================] - 13s 193ms/step - loss: 0.1814
- accuracy: 0.9733 - val_loss: 0.1881 - val_accuracy: 0.9757
Epoch 6/10
68/68 [==============================] - 13s 195ms/step - loss: 0.1483
- accuracy: 0.9733 - val_loss: 0.1720 - val_accuracy: 0.9759
Epoch 7/10
68/68 [==============================] - 14s 202ms/step - loss: 0.1221
- accuracy: 0.9730 - val_loss: 0.1335 - val_accuracy: 0.9753
Epoch 8/10
68/68 [==============================] - 13s 193ms/step - loss: 0.1020
- accuracy: 0.9733 - val_loss: 0.1319 - val_accuracy: 0.9754
Epoch 9/10
68/68 [==============================] - 11s 162ms/step - loss: 0.0863
- accuracy: 0.9738 - val_loss: 0.1747 - val_accuracy: 0.9749
Epoch 10/10
68/68 [==============================] - 13s 186ms/step - loss: 0.0742
- accuracy: 0.9745 - val_loss: 0.1498 - val_accuracy: 0.9744
```

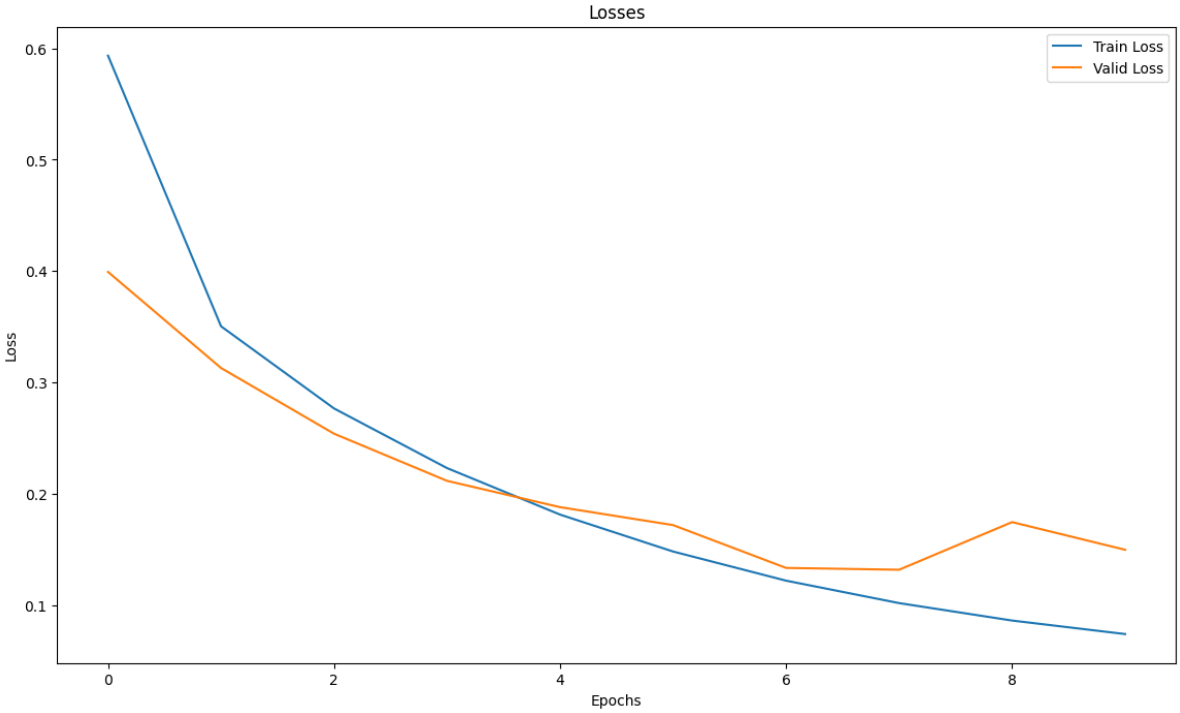
In [27]:

```
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 90634), started 0:10:04 ago. (Us
e '!kill 90634' to kill it.)

<IPython.core.display.Javascript object>
```

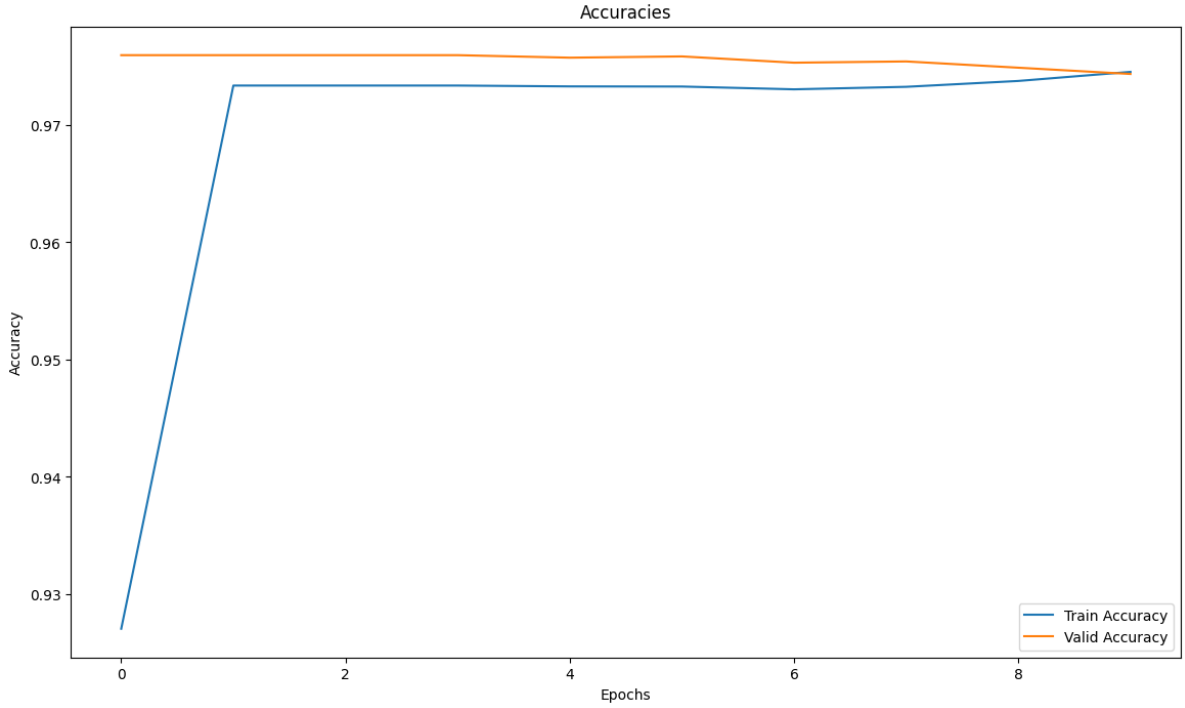
In [28]:

```
plt.figure(figsize=(14,8))
plt.title('Losses')
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Valid Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



In [29]:

```
plt.figure(figsize=(14,8))
plt.title('Accuracies')
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Valid Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



bert-base-uncased model

In [30]:

```
!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.
10/dist-packages (4.31.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/d
ist-packages (from transformers) (3.12.2)
Requirement already satisfied: huggingface-hub<1.0,>=0.14.1 in /usr/lo
cal/lib/python3.10/dist-packages (from transformers) (0.16.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.1
0/dist-packages (from transformers) (1.22.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/pytho
n3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.1
0/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/pyt
hon3.10/dist-packages (from transformers) (2022.10.31)
Requirement already satisfied: requests in /usr/local/lib/python3.10/d
ist-packages (from transformers) (2.27.1)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /u
sr/local/lib/python3.10/dist-packages (from transformers) (0.13.3)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/py
thon3.10/dist-packages (from transformers) (0.3.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.1
0/dist-packages (from transformers) (4.65.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dis
t-packages (from huggingface-hub<1.0,>=0.14.1->transformers) (2023.6.
0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/loca
l/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.14.1->tra
nsformers) (4.7.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/li
b/python3.10/dist-packages (from requests->transformers) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/py
thon3.10/dist-packages (from requests->transformers) (2023.5.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/loca
l/lib/python3.10/dist-packages (from requests->transformers) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.
10/dist-packages (from requests->transformers) (3.4)
```

In [31]:

```
from transformers import AutoConfig, TFAutoModelForTokenClassification

MODEL_NAME = 'bert-base-cased'

config = AutoConfig.from_pretrained(MODEL_NAME, num_labels=len(schema))
model = TFAutoModelForTokenClassification.from_pretrained(MODEL_NAME,
                                                          config=config)
model.summary()
```

```
All PyTorch model weights were used when initializing TFBertForTokenCl
assification.

Some weights or buffers of the TF 2.0 model TFBertForTokenClassificati
on were not initialized from the PyTorch model and are newly initializ
ed: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

Model: "tf_bert_for_token_classification"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bert (TFBertMainLayer)      multiple                  107719680

 dropout_37 (Dropout)        multiple                  0

 classifier (Dense)          multiple                  16918

=================================================================
Total params: 107,736,598
Trainable params: 107,736,598
Non-trainable params: 0
_________________________________________________________________
```

In [32]:

```
from transformers import AutoTokenizer
from tqdm import tqdm
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

def tokenize_sample(sample):
    seq = [
                (subtoken, tag)
                for token, tag in sample
                for subtoken in tokenizer(token)['input_ids'][1:-1]
            ]
    return [(3, 'O')] + seq + [(4, 'O')]

def preprocess(samples):
    tag_index = {tag: i for i, tag in enumerate(schema)}
    tokenized_samples = list(tqdm(map(tokenize_sample, samples)))
    max_len = max(map(len, tokenized_samples))
    X = np.zeros((len(samples), max_len), dtype=np.int32)
    y = np.zeros((len(samples), max_len), dtype=np.int32)
    for i, sentence in enumerate(tokenized_samples):
        for j, (subtoken_id, tag) in enumerate(sentence):
            X[i, j] = subtoken_id
            y[i,j] = tag_index[tag]
    return X, y
```

In [33]:

```
#Train-Test split
X, y = preprocess(train_samples)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.1)
X_test, y_test = preprocess(test_samples)
```

```
2394it [00:04, 578.73it/s]
3850it [00:08, 468.48it/s]
```

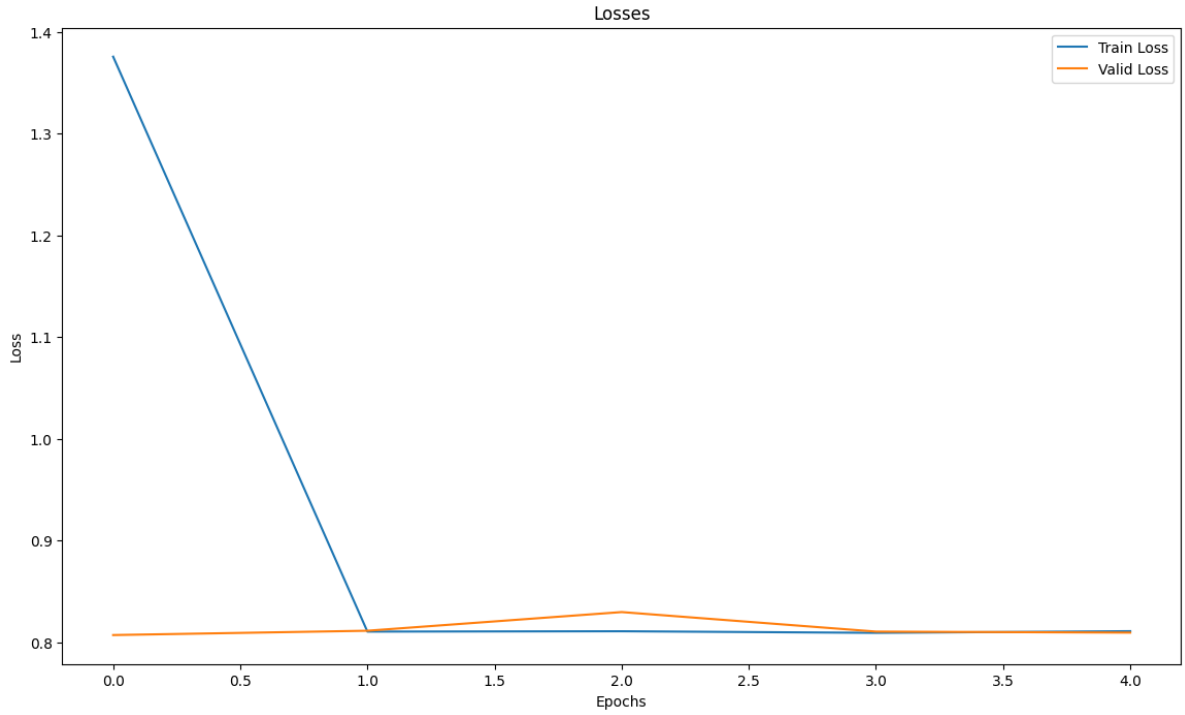
In [34]:

```
EPOCHS=5
BATCH_SIZE=32

optimizer = tf.keras.optimizers.Adam(lr=0.000001)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics='accuracy')
history = model.fit(tf.constant(X_train), tf.constant(y_train),
                    validation_data=(X_val, y_val),
                    epochs=EPOCHS,
                    batch_size=BATCH_SIZE)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learni
ng_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.
Adam.

Epoch 1/5
68/68 [==============================] - 2266s 32s/step - loss: 1.3756
- accuracy: 0.5421 - val_loss: 0.8070 - val_accuracy: 0.6055
Epoch 2/5
68/68 [==============================] - 2220s 33s/step - loss: 0.8104
- accuracy: 0.6076 - val_loss: 0.8112 - val_accuracy: 0.6055
Epoch 3/5
68/68 [==============================] - 2198s 32s/step - loss: 0.8107
- accuracy: 0.6077 - val_loss: 0.8296 - val_accuracy: 0.6055
Epoch 4/5
68/68 [==============================] - 2226s 33s/step - loss: 0.8091
- accuracy: 0.6083 - val_loss: 0.8104 - val_accuracy: 0.6055
Epoch 5/5
68/68 [==============================] - 2200s 32s/step - loss: 0.8108
- accuracy: 0.6095 - val_loss: 0.8094 - val_accuracy: 0.6055
```

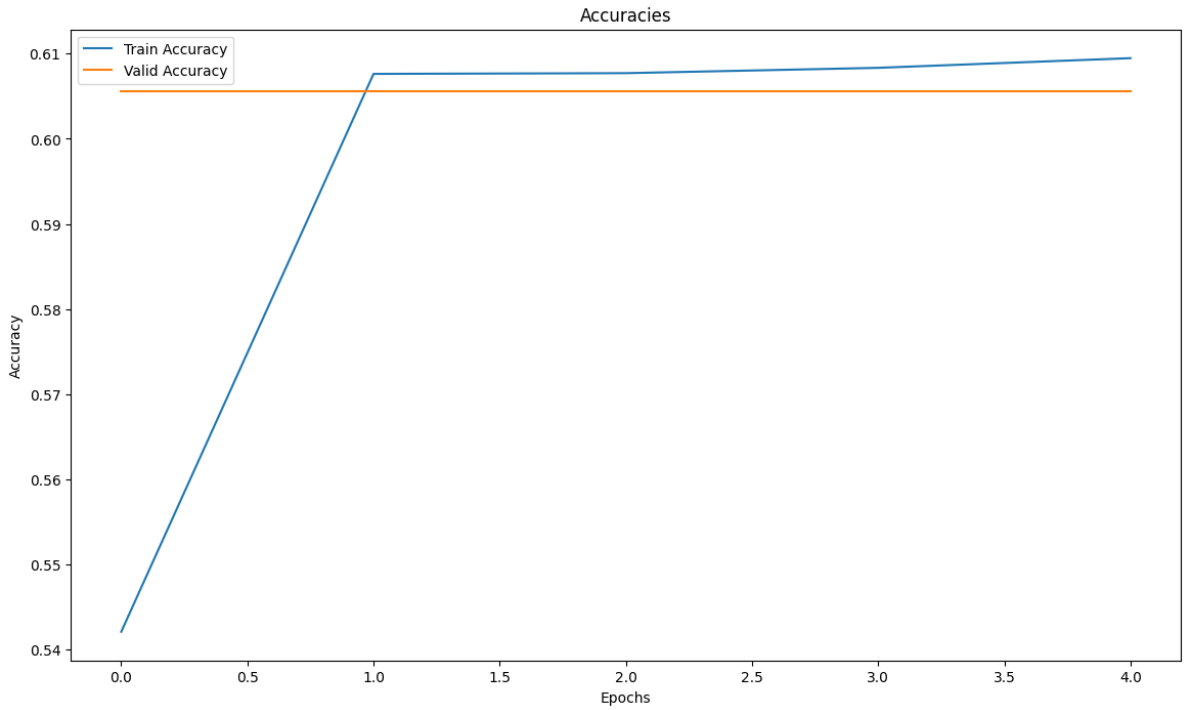
In [35]:

```
plt.figure(figsize=(14,8))
plt.title('Losses')
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Valid Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



In [36]:

```
plt.figure(figsize=(14,8))
plt.title('Accuracies')
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Valid Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```



In [37]:

```
[loss, accuracy] = model.evaluate(X_test, y_test)
print("Loss:%1.3f, Accuracy:%1.3f" % (loss, accuracy))
```

```
121/121 [==============================] - 2572s 21s/step - loss: 0.68
46 - accuracy: 0.7737
Loss:0.685, Accuracy:0.774
```

In [ ]: