Detecting Spam Emails Using Tensorflow in Python

CODE:

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

import string
import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud
nltk.download('stopwords')

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

import warnings
warnings.filterwarnings('ignore')
data = pd.read_csv('/content/archive (4).zip')
data.head()
sns.countplot(x='label', data=data)
plt.show()
ham_msg = data[data['label'] == 'ham']
spam_msg = data[data['label'] == 'spam']

# Downsample Ham emails to match the number of Spam emails
ham_msg_balanced = ham_msg.sample(n=len(spam_msg), random_state=42)
```

```
# Combine balanced data
balanced_data = pd.concat([ham_msg_balanced, spam_msg]).reset_index(drop=True)

# Visualize the balanced dataset
sns.countplot(x='label', data=balanced_data)
plt.title("Balanced Distribution of Spam and Ham Emails")
plt.xticks(ticks=[0, 1], labels=['Ham (Not Spam)', 'Spam'])
plt.show()
balanced_data['text'] = balanced_data['text'].str.replace('Subject', '')
balanced_data.head()
punctuations_list = string.punctuation
def remove_punctuations(text):
    temp = str.maketrans('', '', punctuations_list)
    return text.translate(temp)

balanced_data['text']= balanced_data['text'].apply(lambda x: remove_punctuations(x))
balanced_data.head()
punctuations_list = string.punctuation
def remove_punctuations(text):
    temp = str.maketrans('', '', punctuations_list)
    return text.translate(temp)

balanced_data['text']= balanced_data['text'].apply(lambda x: remove_punctuations(x))
balanced_data.head()
def remove_stopwords(text):
    stop_words = stopwords.words('english')

    imp_words = []

    # Storing the important words
    for word in str(text).split():
```

```
        word = word.lower()

        if word not in stop_words:
            imp_words.append(word)

    output = " ".join(imp_words)

    return output

balanced_data['text'] = balanced_data['text'].apply(lambda text: remove_stopwords(text))
balanced_data.head()
def plot_word_cloud(data, typ):
    email_corpus = " ".join(data['text'])
    wc = WordCloud(background_color='black', max_words=100, width=800,
height=400).generate(email_corpus)
    plt.figure(figsize=(7, 7))
    plt.imshow(wc, interpolation='bilinear')
    plt.title(f'WordCloud for {typ} Emails', fontsize=15)
    plt.axis('off')
    plt.show()

plot_word_cloud(balanced_data[balanced_data['label'] == 'ham'], typ='Non-Spam')
plot_word_cloud(balanced_data[balanced_data['label'] == 'spam'], typ='Spam')
train_X, test_X, train_Y, test_Y = train_test_split(
    balanced_data['text'], balanced_data['label'], test_size=0.2, random_state=42
)

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_X)

train_sequences = tokenizer.texts_to_sequences(train_X)
test_sequences = tokenizer.texts_to_sequences(test_X)
```

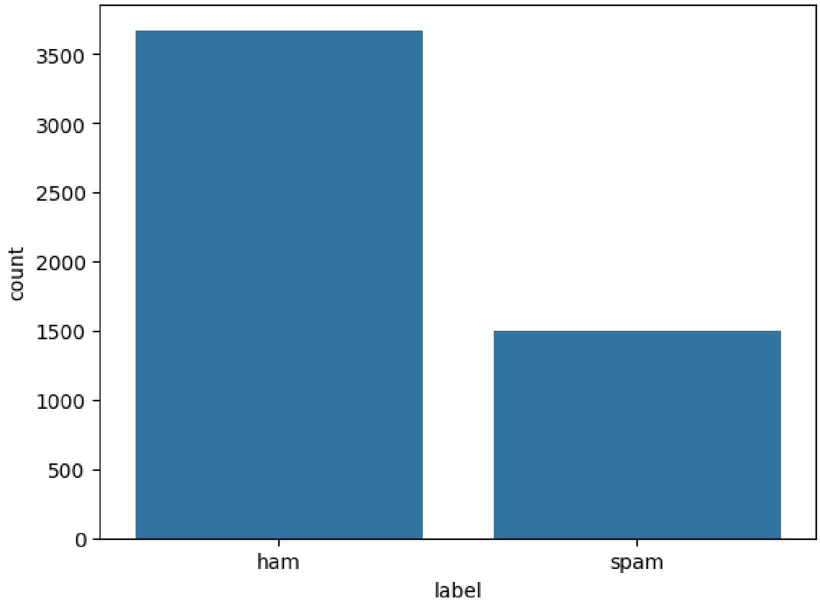
```
max_len = 100  # Maximum sequence length
train_sequences = pad_sequences(train_sequences, maxlen=max_len, padding='post',
truncating='post')
test_sequences = pad_sequences(test_sequences, maxlen=max_len, padding='post', truncating='post')


train_Y = (train_Y == 'spam').astype(int)
test_Y = (test_Y == 'spam').astype(int)
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=32,
input_length=max_len),
    tf.keras.layers.LSTM(16),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Output layer
])


model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    optimizer='adam',
    metrics=['accuracy']
)


model.summary()
es = EarlyStopping(patience=3, monitor='val_accuracy', restore_best_weights=True)
lr = ReduceLROnPlateau(patience=2, monitor='val_loss', factor=0.5, verbose=0)


history = model.fit(
    train_sequences, train_Y,
    validation_data=(test_sequences, test_Y),
    epochs=20,
    batch_size=32,
    callbacks=[lr, es]
)
test_loss, test_accuracy = model.evaluate(test_sequences, test_Y)
```

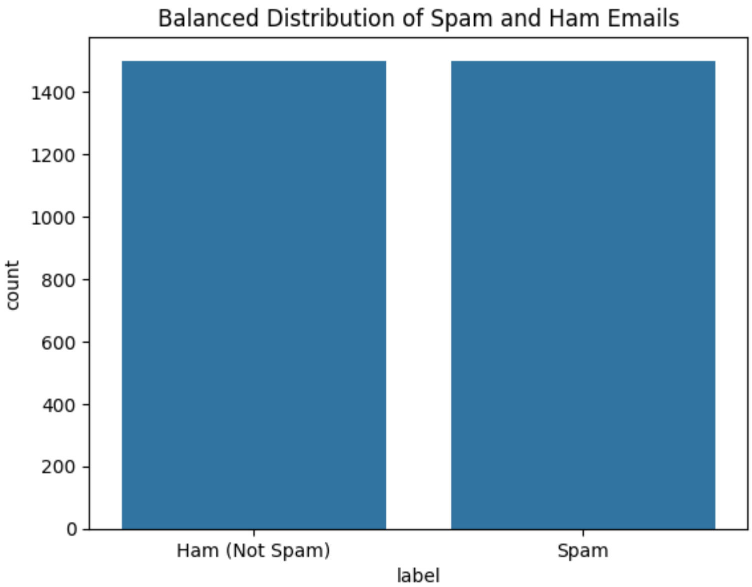
```
print('Test Loss :',test_loss)
print('Test Accuracy :',test_accuracy)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

Output :

[nltk_data] Downloading package stopwords to /root/nltk_data...

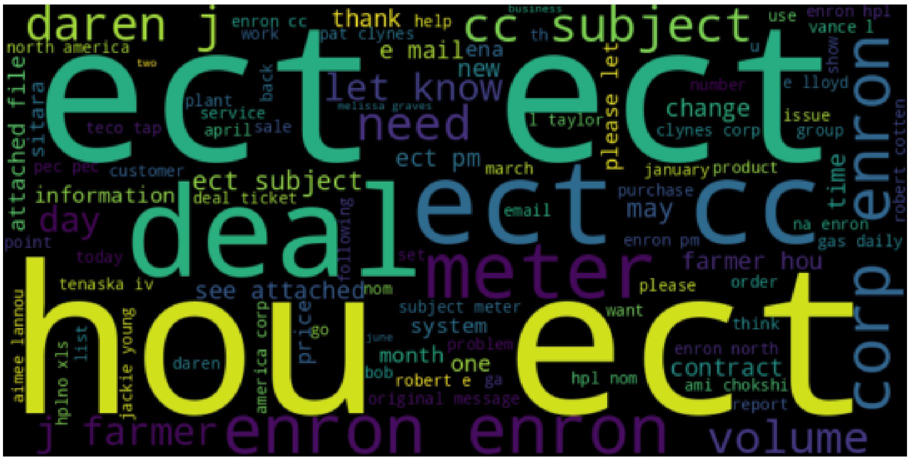
[nltk_data]   Unzipping corpora/stopwords.zip.





	Unnamed: 0	label	text	label_num
0	3444	ham	: conoco - big cowboy\r\ndarren :\r\ni ' m not...	0
1	2982	ham	: feb 01 prod : sale to teco gas processing\r\...	0
2	2711	ham	: california energy crisis\r\ncalifornia □ , s...	0
3	3116	ham	: re : nom / actual volume for april 23 rd\r\n...	0
4	1314	ham	: eastrans nomination changes effective 8 / 2 ...	0

WordCloud for Non-Spam Emails



WordCloud for Spam Emails

Model: "sequential"

Layer (type) Output Shape Param #

===

embedding (Embedding) (None, 100, 32) 1274912 *lstm (LSTM) (None, 16) 3136*

dense (Dense) (None, 32) 544

dense_1 (Dense) (None, 1) 33

===

Total params: 1,278,625
Trainable params: 1,278,625

```
Epoch 1/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 8s 62ms/step - accuracy: 0.5564 - loss: 0.6823 - val_accuracy: 0.9483 - val_loss: 0.2895 - learning_rate: 0.0010
Epoch 2/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 3s 44ms/step - accuracy: 0.9517 - loss: 0.2152 - val_accuracy: 0.9617 - val_loss: 0.1588 - learning_rate: 0.0010
Epoch 3/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - accuracy: 0.9705 - loss: 0.1300 - val_accuracy: 0.9617 - val_loss: 0.1608 - learning_rate: 0.0010
Epoch 4/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 45ms/step - accuracy: 0.9738 - loss: 0.1157 - val_accuracy: 0.9633 - val_loss: 0.1583 - learning_rate: 0.0010
Epoch 5/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step - accuracy: 0.9810 - loss: 0.0908 - val_accuracy: 0.9617 - val_loss: 0.1651 - learning_rate: 0.0010
Epoch 6/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 61ms/step - accuracy: 0.9674 - loss: 0.1330 - val_accuracy: 0.9283 - val_loss: 0.2129 - learning_rate: 0.0010
Epoch 7/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - accuracy: 0.9347 - loss: 0.1965 - val_accuracy: 0.9700 - val_loss: 0.1202 - learning_rate: 5.0000e-04
Epoch 8/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9745 - loss: 0.1062 - val_accuracy: 0.9700 - val_loss: 0.1374 - learning_rate: 5.0000e-04
Epoch 9/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.9879 - loss: 0.0602 - val_accuracy: 0.9683 - val_loss: 0.1481 - learning_rate: 5.0000e-04
Epoch 10/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - accuracy: 0.9883 - loss: 0.0587 - val_accuracy: 0.9683 - val_loss: 0.1495 - learning_rate: 2.5000e-04
```