Detecting Spam Emails Using Tensorflow in Python

Read Discuss Courses Practice

Spam messages refer to unsolicited or unwanted messages/emails that are sent in bulk to users. In most messaging/emailing services, messages are detected as spam automatically so that these messages do not unnecessarily flood the users' inboxes. These messages are usually promotional and peculiar in nature. Thus, it is possible for us to build ML/DL models that can detect Spam messages.

Detecting Spam Emails Using Tensorflow in Python

In this article, we'll build a TensorFlow-based Spam detector; in simpler terms, we will have to classify the texts as **Spam** or **Ham.** This implies that Spam detection is a case of a **Text Classification** problem. So, we'll be performing EDA on our dataset and building a text classification model.

Importing Libraries

Python libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

- **Pandas** – This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
- **Numpy** – Numpy arrays are very fast and can perform large computations in a very short time.

Python3

```
# Importing necessary libraries for EDA
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')

# Importing libraries necessary for Model Building and Training
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')
```

Loading Dataset

Now let's load the dataset into a pandas data frame and look at the first five rows of the dataset. Dataset link – [Email]

Python3

```
data = pd.read_csv('Emails.csv')
data.head()
```

Output:

	text	spam
0	Subject: naturally irresistible your corporate...	1
1	Subject: the stock trading gunslinger fanny i...	1
2	Subject: unbelievable new homes made easy im ...	1
3	Subject: 4 color printing special request add...	1
4	Subject: do not have money , get software cds ...	1

First five rows of the dataset

To check how many such tweets data we have let's print the shape of the data frame.

Python3

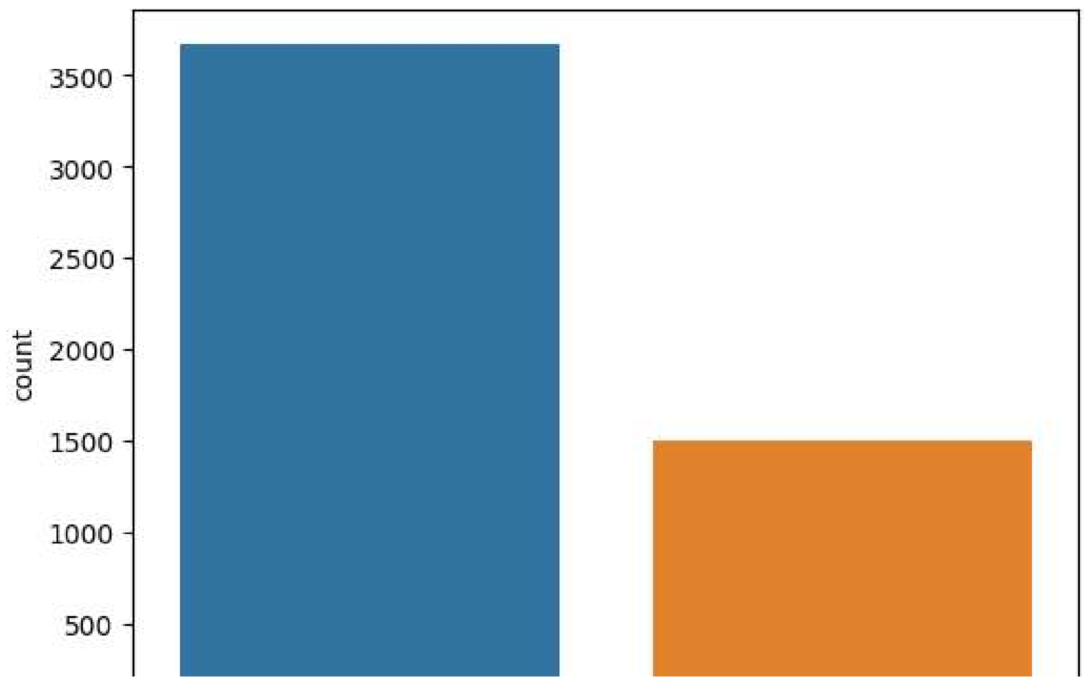
Output:

For a better understanding, we'll plot these counts:

Python3

```
sns.countplot(x='spam', data=data)
plt.show()
```

Output:



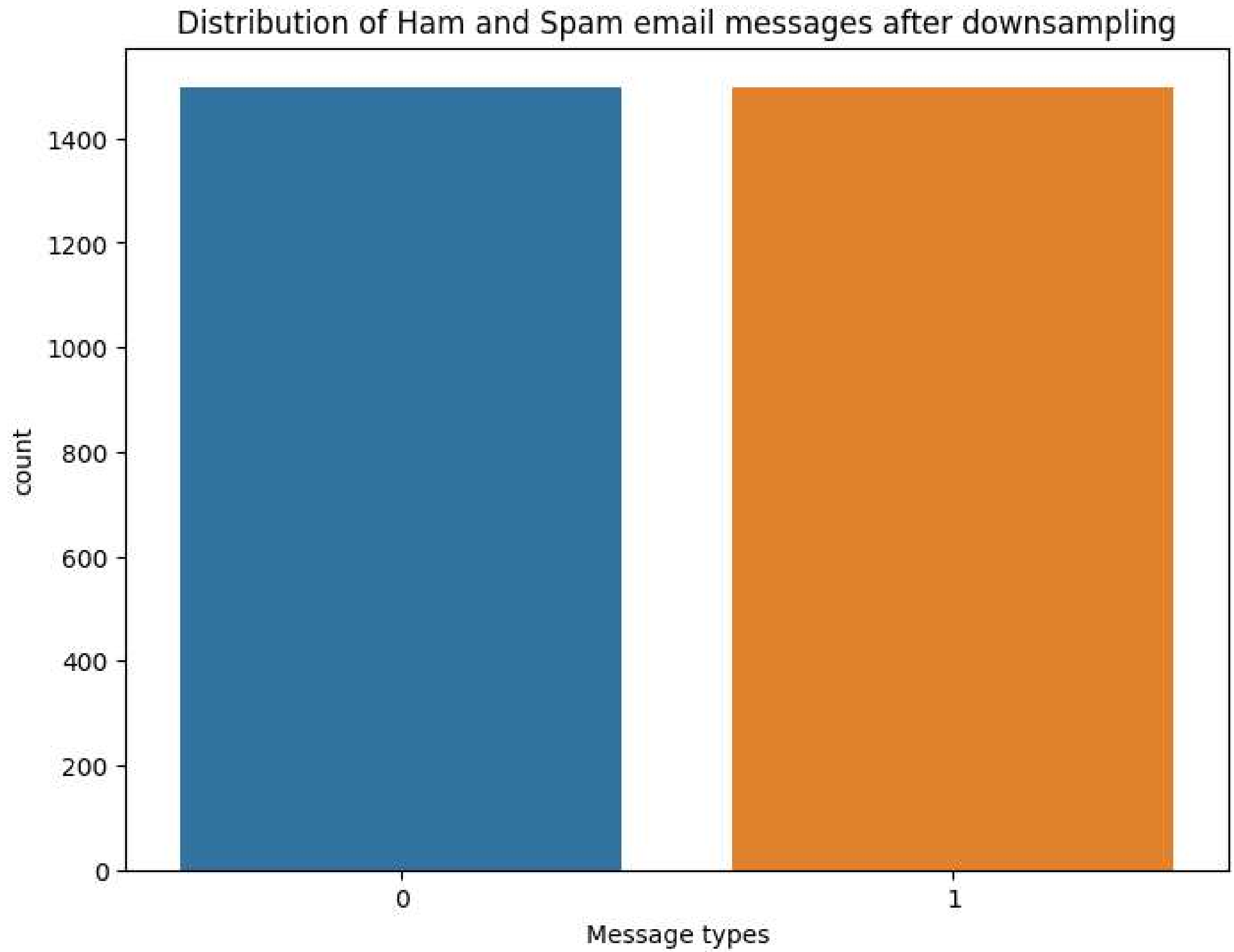
Count plot for the spam labels

We can clearly see that number of samples of Ham is much more than that of Spam which implies that the dataset we are using is imbalanced.

Python3

```
# Downsampling to balance the dataset
ham_msg = data[data.spam == 0]
spam_msg = data[data.spam == 1]
ham_msg = ham_msg.sample(n=len(spam_msg),
                         random_state=42)

# Plotting the counts of down sampled dataset
balanced_data = ham_msg.append(spam_msg)\
    .reset_index(drop=True)
plt.figure(figsize=(8, 6))
sns.countplot(data = balanced_data, x='spam')
plt.title('Distribution of Ham and Spam email messages after downsampling')
plt.xlabel('Message types')
```

Output:



Distribution of Ham and Spam email messages after downsampling

Text Preprocessing

Textual data is highly unstructured and need attention in many aspects:

- Stopwords Removal

Although removing data means loss of information we need to do this to make the data perfect to feed into a machine learning model.

Python3

```
balanced_data['text'] = balanced_data['text'].str.replace('Subject', '')
balanced_data.head()
```

Output:

	Text	Spam
0	: conoco – big cowboy\r\ndarren :\r\ni ‘ m not...	0
1	: feb 01 prod: sale to teco gas processing\r\...	0
2	: california energy crisis\r\ncalifornia , s...	0
3	: re : nom / actual volume for april 23 rd\r\n...	0
4	: eastrans nomination changes effective 8 / 2 ...	0

```
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()
```

Output:

	Text	Spam
0	conoco big cowboy Darren sure helps know else a...	0
1	Feb 01 prod sale teco gas processing sale deal...	0
2	California energy crisis California power cr...	0
3	nom actual volume April 23 rd agree eileen pon...	0
4	eastrans nomination changes effective 8 2 00 p...	0

The below function is a helper function that will help us to remove the stop words.

Python3

```
        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()
```

Output:

	text	spam
0	conoco big cowboy darren sure helps know else a...	0
1	feb 01 prod sale teco gas processing sale deal...	0
2	california energy crisis california power cr...	0
3	nom actual volume April 23rd agree eileen pon...	0

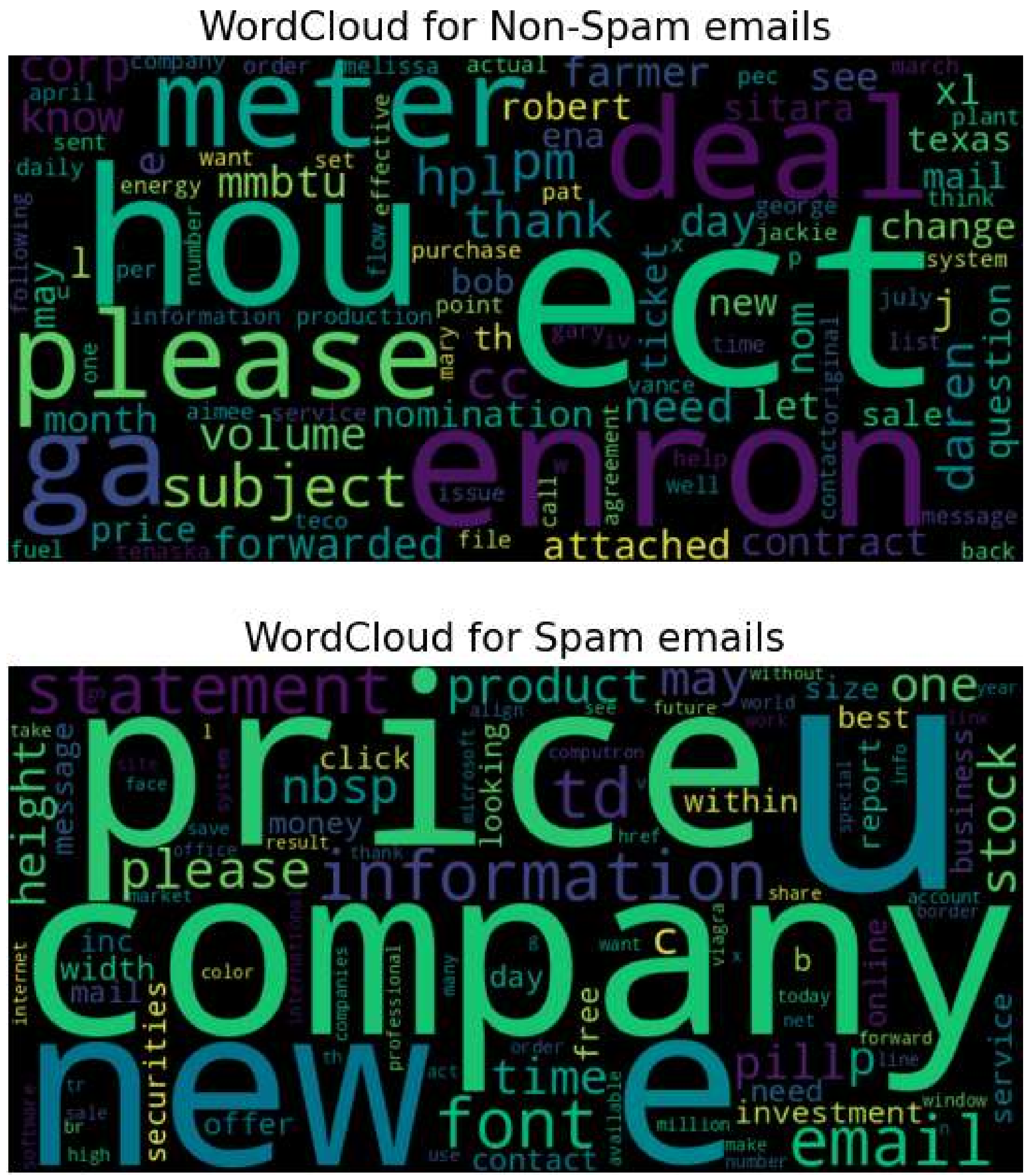
	text	spam
4	eastrans nomination changes effective 8 2 00 p...	0

A word cloud is a text visualization tool that help's us to get insights into the most frequent words present in the corpus of the data.

Python3

```
def plot_word_cloud(data, typ):
    email_corpus = " ".join(data['text'])

    plt.figure(figsize=(7, 7))

    wc = WordCloud(background_color='black',
                   max_words=100,
                   width=800,
                   height=400,
                   collocations=False).generate(email_corpus)

    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['spam'] == 0], typ='Non-Spam')
plot_word_cloud(balanced_data[balanced_data['spam'] == 1], typ='Spam')
```

Output:


WordCloud for Non-Spam emails
meter
deal
hou
ect
please
enron
ga
subject
WordCloud for Spam emails
price
u
company
new
e
information
statement
product

Wordcloud

data will arrive to a stage where we can feed it to a model.

Python3

```
#train test split
train_X, test_X, train_Y, test_Y = train_test_split(balanced_data['text'],
                                                    balanced_data['spam'],
                                                    test_size = 0.2,
                                                    random_state = 42)
```

We have fitted the tokenizer on our training data we will use it to convert the training and validation data both to vectors.

Python3

```
# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_X)

# Convert text to sequences
train_sequences = tokenizer.texts_to_sequences(train_X)
test_sequences = tokenizer.texts_to_sequences(test_X)

# Pad sequences to have the same length
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',
```

Model Development and Evaluation

We will implement a **Sequential model** which will contain the following parts:

- Three Embedding Layers to learn featured vector representations of the input vectors.
- An LSTM layer to identify useful patterns in the sequence.
- Then we will have one fully connected layer.
- The final layer is the output layer which outputs probabilities for the two classes.

Python3

```
# Build the model
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Embedding(input_dim=len(tokenizer.word_index) + 1,
                                    output_dim=32,
                                    input_length=max_len))
model.add(tf.keras.layers.LSTM(16))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

# Print the model summary
model.summary()
```

Output:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 dense (Dense)               (None, 32)                544

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,278,625
Trainable params: 1,278,625
Non-trainable params: 0
_________________________________________________________________
```

While compiling a model we provide these three essential parameters:

- **optimizer** – This is the method that helps to optimize the cost function by using gradient descent.
- **loss** – The loss function by which we monitor whether the model is improving with training or not.
- **metrics** – This helps to evaluate the model by predicting the training and the validation data.

Python3

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

Callback

Callbacks are used to check whether the model is improving with each epoch or not. If not then what are the

Python3

```
es = EarlyStopping(patience=3,
                   monitor = 'val_accuracy',
                   restore_best_weights = True)

lr = ReduceLROnPlateau(patience = 2,
                       monitor = 'val_loss',
                       factor = 0.5,
                       verbose = 0)
```

Let us now train the model:

Python3

```
# Train the model
history = model.fit(train_sequences, train_Y,
                    validation_data=(test_sequences, test_Y),
                    epochs=20,
                    batch_size=32,
                    callbacks = [lr, es]
                   )
```

Output:

```
Epoch 1/20
75/75 [==============================] - 6s 48ms/step - loss: 0.6857 - accuracy: 0.5513 -
val_loss: 0.6159 - val_accuracy: 0.7300 - lr: 0.0010
Epoch 2/20
```

```
75/75 [==============================] - 3s 38ms/step - loss: 0.1590 - accuracy: 0.9625 -
val_loss: 0.1607 - val_accuracy: 0.9600 - lr: 0.0010
Epoch 4/20
75/75 [==============================] - 4s 47ms/step - loss: 0.1856 - accuracy: 0.9545 -
val_loss: 0.1398 - val_accuracy: 0.9700 - lr: 0.0010
Epoch 5/20
75/75 [==============================] - 3s 43ms/step - loss: 0.0781 - accuracy: 0.9850 -
val_loss: 0.1122 - val_accuracy: 0.9750 - lr: 0.0010
Epoch 6/20
75/75 [==============================] - 3s 46ms/step - loss: 0.0563 - accuracy: 0.9908 -
val_loss: 0.1129 - val_accuracy: 0.9767 - lr: 0.0010
Epoch 7/20
75/75 [==============================] - 3s 42ms/step - loss: 0.0395 - accuracy: 0.9937 -
val_loss: 0.1088 - val_accuracy: 0.9783 - lr: 0.0010
Epoch 8/20
75/75 [==============================] - 4s 50ms/step - loss: 0.0327 - accuracy: 0.9950 -
val_loss: 0.1303 - val_accuracy: 0.9750 - lr: 0.0010
Epoch 9/20
75/75 [==============================] - 3s 43ms/step - loss: 0.0272 - accuracy: 0.9958 -
val_loss: 0.1337 - val_accuracy: 0.9750 - lr: 0.0010
Epoch 10/20
75/75 [==============================] - 3s 43ms/step - loss: 0.0247 - accuracy: 0.9962 -
val_loss: 0.1351 - val_accuracy: 0.9750 - lr: 5.0000e-04
```

Now, let's evaluate the model on the validation data.

```
test_loss, test_accuracy = model.evaluate(test_sequences, test_Y)
print('Test Loss :',test_loss)
print('Test Accuracy :',test_accuracy)
```

Output:

```
19/19 [==============================] - 0s 7ms/step - loss: 0.1088 - accuracy: 0.9783
Test Loss : 0.1087912991642952
Test Accuracy : 0.9783333539962769
```

Thus, the training accuracy turns out to be 97.44% which is quite satisfactory.

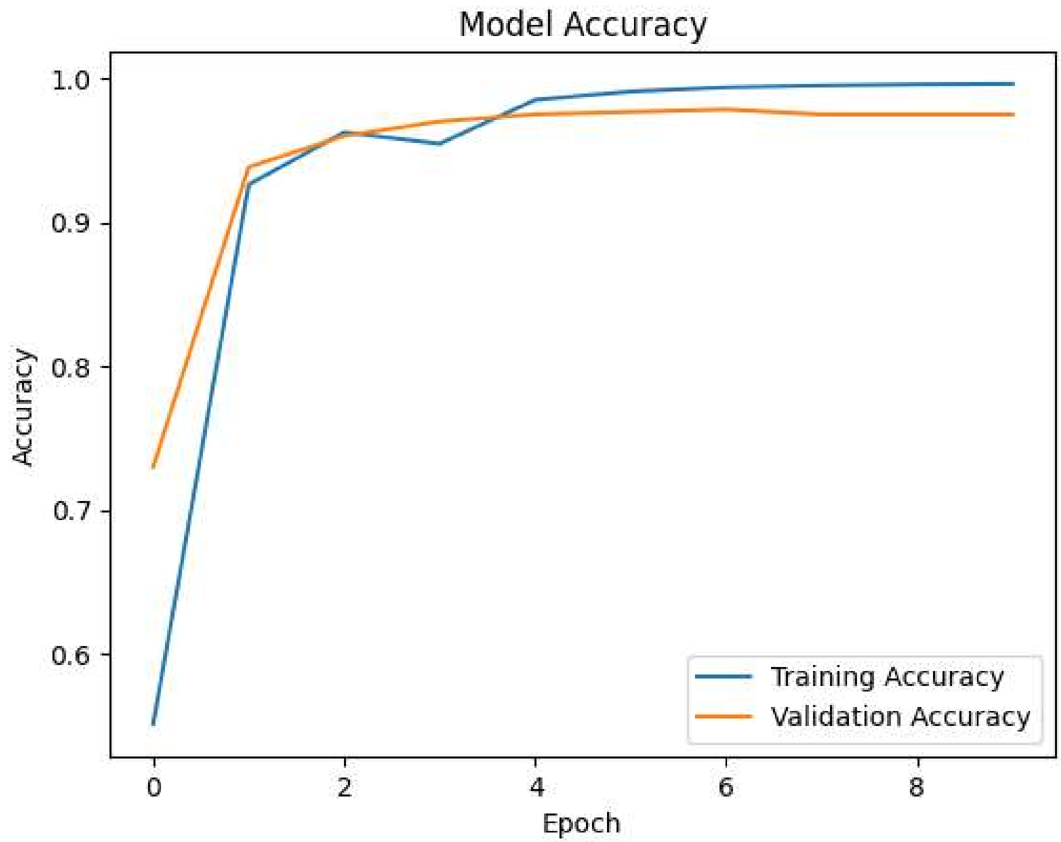
Model Evaluation Results

Having trained our model, we can plot a graph depicting the variance of training and validation accuracies with the no. of epochs.

Python3

```
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:



Model Accuracy

Last Updated : 17 May, 2023