```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer



from tensorflow.keras.datasets import imdb

# Load the IMDb dataset
vocab_size = 10000  # Use the top 10,000 words in the dataset
maxlen = 100  # Maximum length of each review (in words)

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocab_size)

# Pad sequences to ensure uniform input size
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
model = Sequential()

# Add Embedding layer to convert words into dense vectors of fixed size
model.add(Embedding(input_dim=vocab_size, output_dim=128, input_length=maxlen))

# Add LSTM layer
model.add(LSTM(units=128, return_sequences=False))

# Add a Dropout layer to reduce overfitting
model.add(Dropout(0.5))

# Add Dense layer with sigmoid activation for binary classification
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Model summary
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. J
  warnings.warn(
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dropout (Dropout)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
```

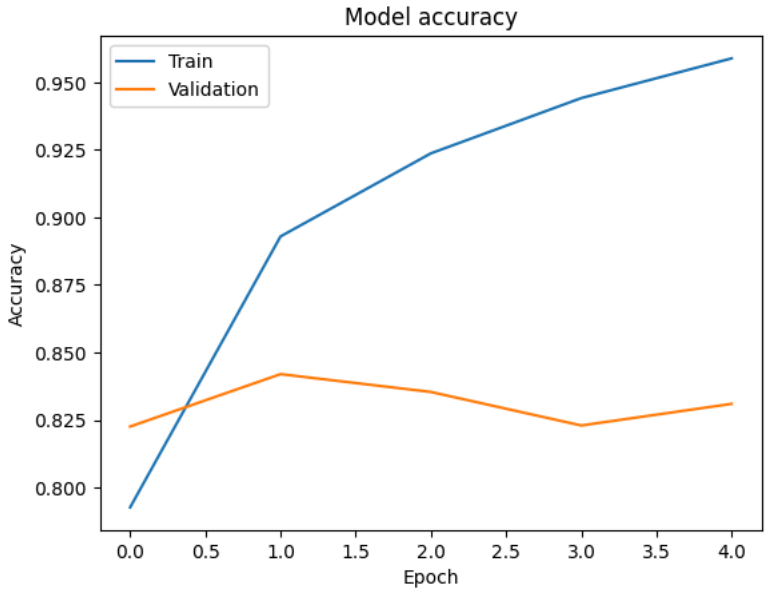
```
# Train the model
batch_size = 64
epochs = 5

history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

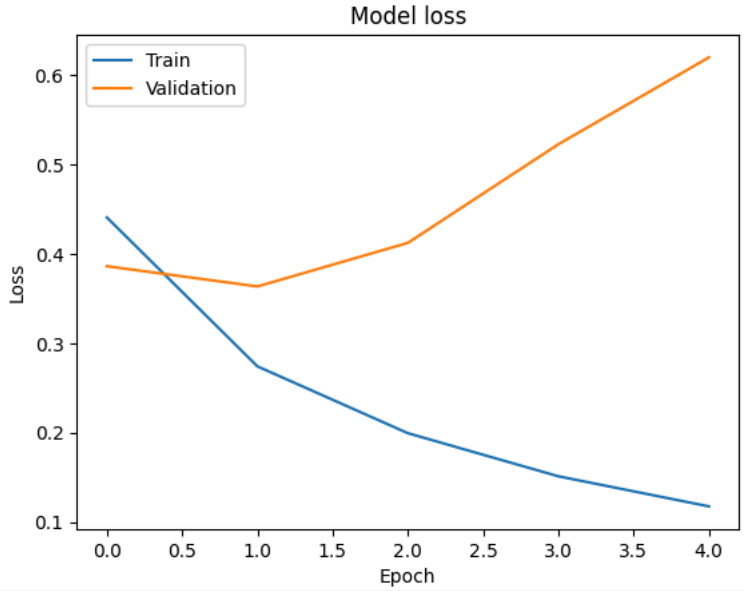
```
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 88s 272ms/step - accuracy: 0.7169 - loss: 0.5329 - val_accuracy: 0.8226 - val_loss: 0.3861
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 141s 269ms/step - accuracy: 0.8948 - loss: 0.2719 - val_accuracy: 0.8420 - val_loss: 0.3635
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 144s 278ms/step - accuracy: 0.9301 - loss: 0.1871 - val_accuracy: 0.8354 - val_loss: 0.4122
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 140s 270ms/step - accuracy: 0.9486 - loss: 0.1387 - val_accuracy: 0.8230 - val_loss: 0.5225
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 86s 276ms/step - accuracy: 0.9637 - loss: 0.1091 - val_accuracy: 0.8310 - val_loss: 0.6196
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Accuracy: {test_accuracy:.2f}")
# Example of making predictions
predictions = model.predict(X_test[:5])
predicted_labels = (predictions > 0.5).astype(int).flatten()

# Compare with true labels
print(f"Predicted labels: {predicted_labels}")
print(f"True labels: {y_test[:5]}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 49s 63ms/step - accuracy: 0.8263 - loss: 0.6449
Test Accuracy: 0.83
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 185ms/step
Predicted labels: [0 1 1 1 1]
True labels: [0 1 1 0 1]
```

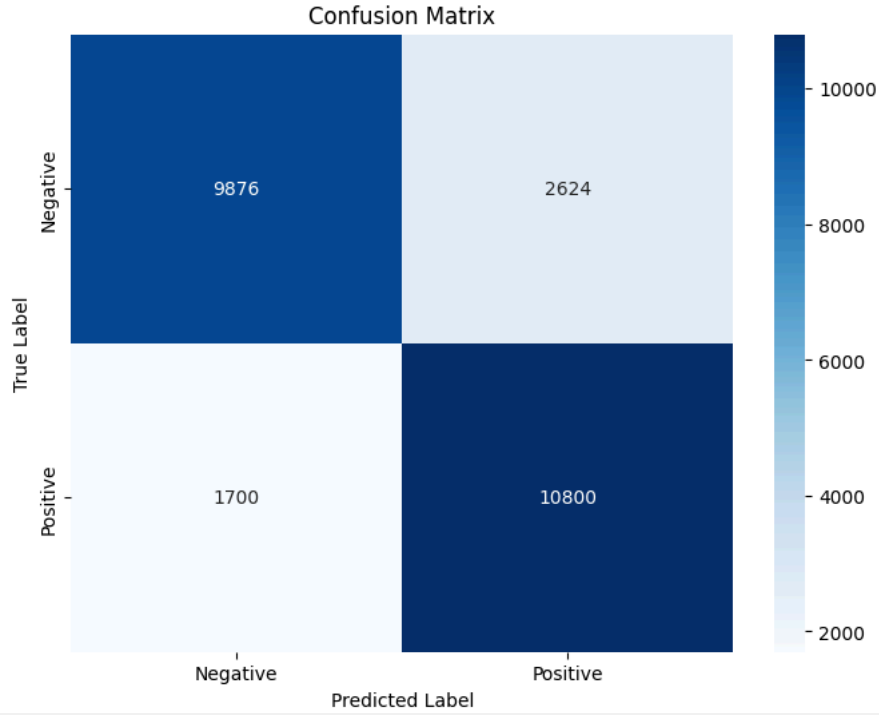
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```






```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Generate predictions for the test set
y_pred_lstm = (model.predict(X_test) > 0.5).astype("int32")

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred_lstm)
print("Confusion Matrix:")
print(cm)

# Generate the classification report
cr = classification_report(y_test, y_pred_lstm)
print("Classification Report:")
print(cr)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Negative", "Positive"], yticklabels=["Negative", "Positive"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 53s 67ms/step
Confusion Matrix:
[[ 9876  2624]
 [ 1700 10800]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.79      0.82     12500
           1       0.80      0.86      0.83     12500

    accuracy                           0.83     25000
   macro avg       0.83      0.83      0.83     25000
weighted avg       0.83      0.83      0.83     25000
```




```
# Train the model
batch_size = 64
epochs = 5

history_rnn = model_rnn.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.2)
```

```
Epoch 1/5
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-13-49e7fe6475ed> in <cell line: 5>()
      3 epochs = 5
      4
----> 5 history_rnn = model_rnn.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.2)

                                 1 frames
/usr/local/lib/python3.10/dist-packages/keras/src/layers/input_spec.py in assert_input_compatibility(input_spec, inputs,
layer_name)
    184         if spec.ndim is not None and not spec.allow_last_axis_squeeze:
    185             if ndim != spec.ndim:
--> 186                 raise ValueError(
    187                     f'Input {input_index} of layer "{layer_name}" '
    188                     "is incompatible with the layer: "

ValueError: Input 0 of layer "simple_rnn" is incompatible with the layer: expected ndim=3, found ndim=4. Full shape received:
```

Next steps: Explain error

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from tensorflow.keras.datasets import imdb
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
```

```
import seaborn as sns

# Load the IMDb dataset
vocab_size = 10000  # Use the top 10,000 words in the dataset
maxlen = 100  # Maximum length of each review (in words)

(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocab_size)

# Pad sequences to ensure uniform input size
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)

# RNN model
model_rnn = Sequential()
model_rnn.add(Embedding(input_dim=vocab_size, output_dim=128, input_length=maxlen))
model_rnn.add(SimpleRNN(units=128, return_sequences=False))
model_rnn.add(Dropout(0.5))
model_rnn.add(Dense(1, activation='sigmoid'))

# Compile the model
model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Model summary
model_rnn.summary()

# Train the model
batch_size = 64
epochs = 5

history_rnn = model_rnn.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.2)

# Evaluate the model on the test set
test_loss_rnn, test_accuracy_rnn = model_rnn.evaluate(X_test, y_test, verbose=1)
print(f"Test Accuracy (RNN): {test_accuracy_rnn:.2f}")

# Generate predictions for the test set
y_pred_rnn = (model_rnn.predict(X_test) > 0.5).astype("int32")

# Generate the confusion matrix
cm_rnn = confusion_matrix(y_test, y_pred_rnn)
print("Confusion Matrix (RNN):")
print(cm_rnn)

# Generate the classification report
cr_rnn = classification_report(y_test, y_pred_rnn)
print("Classification Report (RNN):")
print(cr_rnn)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_rnn, annot=True, fmt="d", cmap="Blues", xticklabels=["Negative", "Positive"], yticklabels=["Negative", "Positive"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix (RNN)")
plt.show()
```

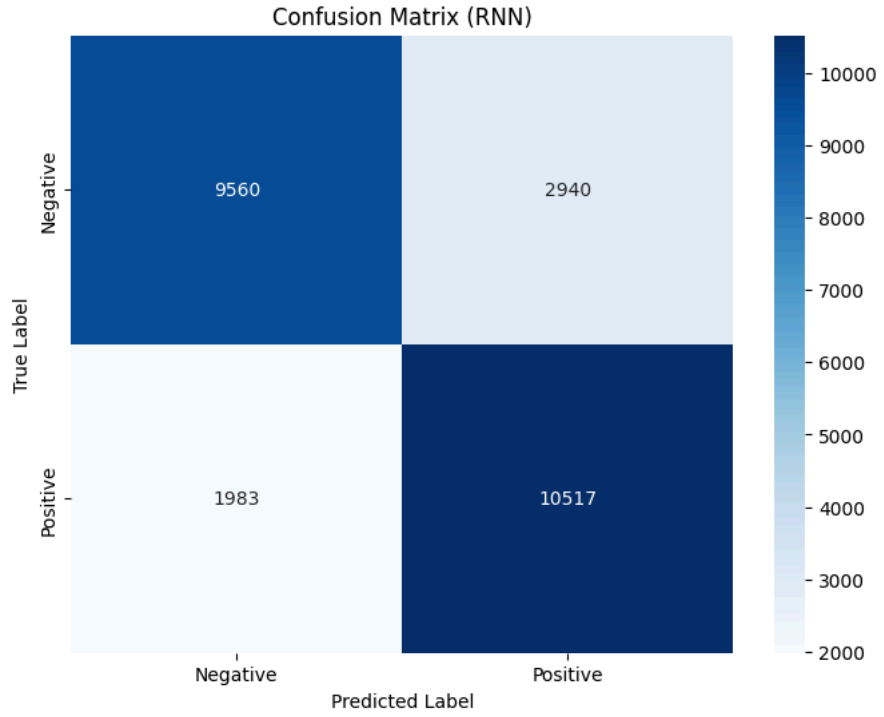
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated
  warnings.warn(
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	?	0 (unbuilt)
simple_rnn_1 (SimpleRNN)	?	0 (unbuilt)
dropout_2 (Dropout)	?	0 (unbuilt)
dense_2 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 27s 77ms/step - accuracy: 0.5310 - loss: 0.6976 - val_accuracy: 0.6240 - val_loss: 0.6373
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 25s 81ms/step - accuracy: 0.7021 - loss: 0.5568 - val_accuracy: 0.7990 - val_loss: 0.4406
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 42s 84ms/step - accuracy: 0.8552 - loss: 0.3443 - val_accuracy: 0.8090 - val_loss: 0.4352
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 27s 86ms/step - accuracy: 0.8862 - loss: 0.2841 - val_accuracy: 0.8130 - val_loss: 0.4727
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 36s 114ms/step - accuracy: 0.9281 - loss: 0.1938 - val_accuracy: 0.8024 - val_loss: 0.5386
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 12ms/step - accuracy: 0.8000 - loss: 0.5298
Test Accuracy (RNN): 0.80
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step
Confusion Matrix (RNN):
[[ 9560  2940]
 [ 1983 10517]]
Classification Report (RNN):
              precision    recall  f1-score   support

           0       0.83      0.76      0.80     12500
           1       0.78      0.84      0.81     12500

    accuracy                           0.80     25000
   macro avg       0.80      0.80      0.80     25000
weighted avg       0.80      0.80      0.80     25000
```



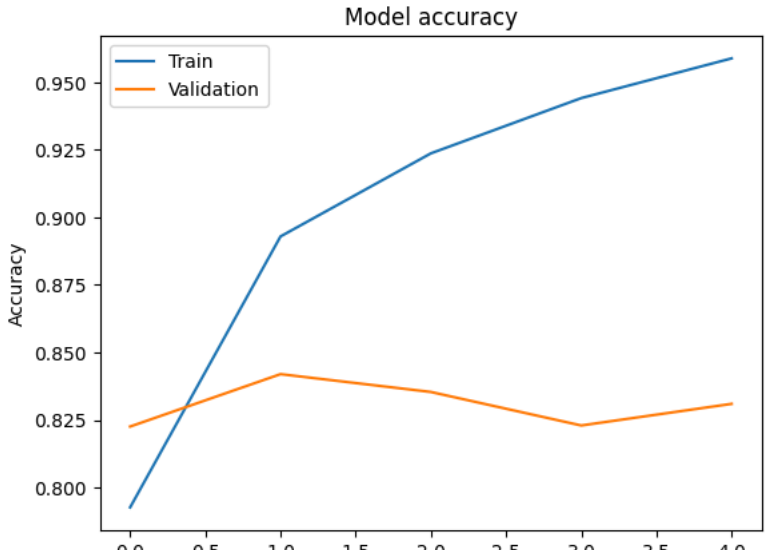

```
# Evaluate the model on the test set
test_loss, test_accuracy = model_rnn.evaluate(X_test, y_test, verbose=1)
print(f"Test Accuracy: {test_accuracy:.2f}")
# Example of making predictions
predictions_rnn = model_rnn.predict(X_test[:5])
predicted_labels_rnn = (predictions > 0.5).astype(int).flatten()

# Compare with true labels
print(f"Predicted labels: {predicted_labels_rnn}")
print(f"True labels: {y_test[:5]}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - accuracy: 0.8000 - loss: 0.5298
Test Accuracy: 0.80
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Predicted labels: [0 1 1 1 1]
True labels: [0 1 1 0 1]
```

```
from sklearn.metrics import confusion_matrix, classification_report

# Generate predictions for the test set
y_pred_rnn = (model.predict(X_test) > 0.5).astype("int32")

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred_rnn)
print("Confusion Matrix:")
print(cm)

# Generate the classification report
cr = classification_report(y_test, y_pred_rnn)
print("Classification Report:")
print(cr)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 46s 59ms/step
Confusion Matrix:
[[ 9876  2624]
 [ 1700 10800]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.79      0.82     12500
           1       0.80      0.86      0.83     12500

    accuracy                           0.83     25000
   macro avg       0.83      0.83      0.83     25000
weighted avg       0.83      0.83      0.83     25000
```

```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```




```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Predictions for LSTM model
y_pred_lstm = (model.predict(X_test) > 0.5).astype("int32")

# Calculate metrics for LSTM model
lstm_accuracy = accuracy_score(y_test, y_pred_lstm)
lstm_precision = precision_score(y_test, y_pred_lstm)
lstm_recall = recall_score(y_test, y_pred_lstm)
lstm_f1 = f1_score(y_test, y_pred_lstm)

# Predictions for SimpleRNN model
y_pred_rnn = (model_rnn.predict(X_test) > 0.5).astype("int32")

# Calculate metrics for SimpleRNN model
rnn_accuracy = accuracy_score(y_test, y_pred_rnn)
rnn_precision = precision_score(y_test, y_pred_rnn)
rnn_recall = recall_score(y_test, y_pred_rnn)
rnn_f1 = f1_score(y_test, y_pred_rnn)

print("LSTM Model Metrics:")
print(f"Accuracy: {lstm_accuracy:.4f}")
print(f"Precision: {lstm_precision:.4f}")
print(f"Recall: {lstm_recall:.4f}")
print(f"F1-Score: {lstm_f1:.4f}")

print("\nSimpleRNN Model Metrics:")
print(f"Accuracy: {rnn_accuracy:.4f}")
print(f"Precision: {rnn_precision:.4f}")
print(f"Recall: {rnn_recall:.4f}")
print(f"F1-Score: {rnn_f1:.4f}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 47s 59ms/step
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step
LSTM Model Metrics:
```